1992-03

Design and specification of a high speed transport protocol

McArthur, Robert C.

Monterey, California. Naval Postgraduate School

Approved for public release; distribution is unlimited

DESIGN AND SPECIFICATION OF A HIGH SPEED TRANSPORT PROTOCOL

by
Robert C. McArthur
Captain, United States Marine Corps
B. S., United States Naval Academy, 1985

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 1992

Robert B. McGhee, Chairman,
Department of Computer Science

REPORT DOCUMENTATION PAGE

REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
SECURITY CLASSIFICATION AUTHORITY DECLASSIFICATION/DOWNGRADING SCHEDULE	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
ERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)

NAME OF PERFORMING ORGANIZATION mputer Science Dept. val Postgraduate School	6b. OFFICE SYMBOL (if applicable) CS	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
ADDRESS (City, State, and ZIP Code) onterey, CA 93943-5000		7b. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000
NAME OF FUNDING/SPONSORING ORGANIZATION	8b. OFFICE SYMBOL (if applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
ADDRESS (City, State, and ZIP Code)		10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNI ACCESSIO

TITLE (Include Security Classification)
SIGN AND SPECIFICATION OF A HIGH SPEED TRANSPORT PROTOCOL (U)

PERSONAL AUTHOR(S)
Arthur, Robert C.

TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
ster's Thesis	FROM 09/91 to 03/92	March 1992	92

SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official icy or position of the Department of Defense or the United States Government.

COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
IELD	GROUP	SUB-GROUP	Transport Protocol, Protocol Specification, Lightweight Transport Proto System of Communicating Machines, Protocol Analysis

ABSTRACT (Continue on reverse if necessary and identify by block number)

Due to the increase in data throughput potential provided by high speed (fiber optic) networks, existing trans otocols are becoming increasingly incapable of providing reliable and timely transfer of data. Whereas in netw the past it was the transmission medium that caused the greatest communications delay, it is the case today tha nsport protocols themselves have become the bottleneck. This thesis provides detailed insight into the issues affecting the development of the next generation of high speed transport protocols, and includes a formal spe ion and limited analysis of one such protocol. Through a dissection of transport protocol functions, this thesi trates some of the problems which are hindering optimal performance, and demonstrates some of the design erations of new transport protocols which are providing significant gains in efficiency. Three of the most pro lightweight transport protocol research projects are surveyed to provide a frame of reference for the ne erging design paradigm: taking advantage of the low error rate of fiber optic media to optimize success, rather mpensate for failure.

DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED □ SAME AS RPT. □ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
NAME OF RESPONSIBLE INDIVIDUAL M. Lundy	22b. TELEPHONE (Include Area Code) (408) 646-2449	22c. OFFICE SYMBOL CS/37

83 APR edition may be used until exhausted
All other editions are obsolete

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

DESIGN AND SPECIFICATION OF A HIGH SPEED TRANSPORT PROTOCOL

by

Captain Robert C. McArthur, USMC

26 March 1992

Thesis Advisor: G.M. Lundy

Approved for public release; distribution is unlimited.

ABSTRACT

Due to the increase in data throughput potential provided by high speed (fiber optic) networks, existing transport protocols are becoming increasingly incapable of providing reliable and timely transfer of data. Whereas in networks of the past it was the transmission medium that caused the greatest communications delay, it is the case today that the transport protocols themselves have become the bottleneck. This thesis provides detailed insight into the issues that are affecting the development of the next generation of high speed transport protocols, and includes a formal specification and limited analysis of one such protocol. Through a dissection of transport protocol functions, this thesis illustrates some of the problems which are hindering optimal performance, and demonstrates some of the design considerations of new transport protocols which are providing significant gains in efficiency. Three of the most promising lightweight transport protocol research projects are surveyed to provide a frame of reference for the newly emerging design paradigm: taking advantage of the low error rate of fiber optic media to optimize success, rather than compensate for failure.

C.1

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

Thank you Denise, for putting up with the long and irregular hours over the past two years. Without your support and assistance in maintaining a focused objective, this would have been a frustrating tour of duty for me. This work is dedicated to you.

Thank you Joshua and Zachary for continually reminding me that there is an innocence to each of us which can provide us with peace, and an insatiable sense of wonder which can lead us to new discovery. I hope that I never forget how to look at the World through the eyes of a child.

Thank you Dr. Lundy for accepting me as a thesis student so late in the program, and for instilling in me an interest in the field of communications.

And finally, thanks to the United States Marine Corps for giving me the opportunity to further my education and professional ability. I will do my best to apply what I have learned during this tour of duty to my future assignments, so that the Marine Corps will continue to maintain its role as the finest fighting force in the history of the World.

Semper Fi.

I. INTRODUCTION

A. BACKGROUND

The introduction of high speed (Gbps) networks into the computer communications industry has brought to light the need for a redesign effort covering implemented transport protocols. The order of magnitude increase in data throughput potential provided by fiber optic media has caused existing transport protocols to become increasingly incapable of providing timely and reliable transfer of data. Whereas in networks of the past the physical limitations of the medium caused a significant percentage of the communication delay, today protocol processing has become the bottleneck. In [NETR90], the authors show that long latency, high speed networks using current transport protocols could experience states in which a transmission or network error or buffer overflow could necessitate retransmission of 60 Mbits of data.

Protocol designers are developing a new generation of transport protocols which focus upon exploiting the high bandwidth, low transmission error capabilities of fiber optics. Most current research refers to these new protocols as *lightweight,* or *high-speed* transport protocols. The approach taken to produce these protocols is quite different from that used in developing past transport protocols in that the design process is centered upon optimizing performance toward successful packet transmission, rather than toward building robustness to compensate for transmission failure.

B. OBJECTIVES

In this thesis, the design considerations for lightweight transport protocols will be investigated in detail. This will include a review of transport protocol functions, and a discussion of the shortcomings of currently implemented transport protocols which makes them unacceptable for use with high-speed networks. A survey of current transport protocol research projects will be presented to illustrate some of the new features which may be integrated into future communication systems.

The major contribution of this thesis is the formal specification of the lightweight transport protocol introduced in [NETR90], and an analysis of that protocol in terms of functional efficiency . Because the formal specification in this thesis is built upon the informal specification presented in that paper, and because many references will be made to the informal specification, it will be referred to throughout this thesis as the *AT&T Specification*. By modeling the protocol using the System of Communicating Machines (SCM) model described in [LUND88], the behavior of the protocol is clearly defined, and its functionality demonstrated.

C. SCOPE

This thesis is a study of the functions and capabilities of lightweight transport protocols, including the formal specification and analysis discussed above. The specification presented lends itself very well to simulation or automated analysis of the protocol in that the communicating machines and processes are well-defined entities which interact through clearly established communication channels [ROTH92]. Performance tests between this and other lightweight transport protocols have not yet been conducted, though [DOER90] does present a very informative comparison of the design features of the most promising research projects.

The formal specification section of this thesis refines the precision of the AT&T Specification by modifying some of the state machines and data structures used to model communication processes. The specification also expands the scope of the AT&T Specification by including the connection establishment and disestablishment functions of the protocol. The AT&T Specification is not repeated here, and a complete summary of modifications is not included. In instances where significant modifications were necessary a discussion of relevant design considerations is included.

An analysis of some specific protocol functions is included to illustrate the use of the SCM model, and to provide a more convenient presentation of the protocol's execution. Included in the analysis is a discussion of some of the features of the protocol which may hinder its performance in some high-speed environments.

The specification does not explicitly present management of a multiplexed connection. From an abstract viewpoint, the specification could be duplicated for every logical connection of a multiplexed port, with a logical connection identifier field used to route incoming packets to the appropriate processing location.

D. ORGANIZATION

This thesis is organized into three primary sections, with topics covered toward the end of the thesis building upon concepts discussed earlier. The first section contains Chapters II and III, which cover general transport protocol functions, and a survey of lightweight protocol research projects. The purpose is to set the stage for the more detailed discussions and protocol specification which follow. The primary responsibilities of a transport protocol are studied, and the requirement for lightweight protocols is made clear.

A general description of the AT&T Transport Protocol is given in Chapter IV, the formal definition of the SCM model in Chapter V, definitions for the specification structures in Chapter VI, and the formal specification of the AT&T Transport Protocol in Chapter VII. Chapters IV through VI are a necessary introduction to the formal specification in that the specification itself does not elaborate upon the details of its processes. Having gained an understanding of the various components of the model and the functionality of the protocol, the reader will find it much easier to follow the formal specification.

Finally, an analysis of some of the key features of the protocol is presented in Chapter VIII, and a conclusion to the thesis is provided in Chapter IX. The results of the research will be discussed, and recommendations for future study will be offered.

II. TRANSPORT PROTOCOLS

A. COMMUNICATIONS ARCHITECTURES

In order to fully understand the function of transport protocols it is necessary to be familiar with the concept of a *communications architecture.* Simply stated, a communications architecture is a set of protocols, each of which *uses the services* of a given subset of the protocols, and *provides services* to another (disjoint) subset. Typically, the protocols in the architecture are layered or hierarchically organized so that there is a continuous flow of responsibility, from establishing connectivity between communicating entities at the lowest level to providing a well integrated user interface at the upper level.

In some architectures a particular protocol may only use the services of the protocol directly preceding it (in terms of functional responsibility), and may only provide services to the protocol directly subsequent to it. The most notable example of this structure is the International Standards Organization (ISO) Open Systems Interface(OSI) model, which requires a seven layered architecture at each communicating entity. Each layer of the "stack" contains a separate protocol which is responsible for performing a specific function relating to transmission or receipt of a data packet. Logically, a given layer within the stack communicates directly with its corresponding layer in another stack. In actuality, any communication between two corresponding layers may only be conducted by passing a data packet through each of the protocol layers preceding them in the stack. Figure 1 shows an abstract view of the communication process in the ISO OSI model.

In other architectures the relationship between the various protocols may not be so ordered. In the late 1970's the Defense Advanced Research Projects Agency (DARPA) constructed a communications architecture which has grown to serve most of the United

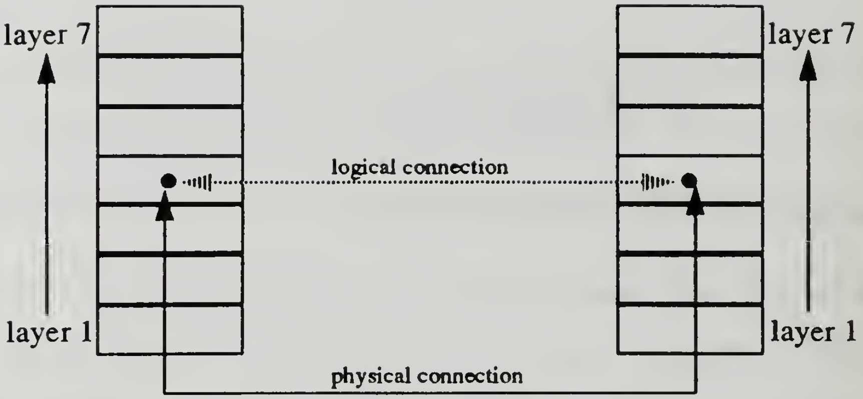


Figure 1 : ISO OSI Communication Architecture Model

States and many foreign countries as well. This architecture is commonly referred to as TCP/IP, and the international network infrastructure which uses this architecture is called “The Internet”.

In TCP/IP, the protocol set is hierarchically organized rather than strictly layered. As a result, a particular protocol may use the services of more than one other protocol in the set, whereas in the OSI model, a particular protocol could only use the services of its predecessor. The term “layer” does not map into the TCP/IP model precisely, though in this thesis it will be used to refer to either an OSI layer or a TCP/IP *functional responsibility*. Figure 2 shows the organization of the TCP/IP suite of protocols [STAL88].

B. TRANSPORT PROTOCOL FUNCTIONS

Within a communications architecture the transport protocol is responsible for ensuring the reliable, end-to-end communication of data and control information. The transport protocol uses the services of the layers “beneath” it , either directly or indirectly, to pass data or control packets across a communication channel or network. In the OSI model, the fourth layer contains the transport protocol (TP4). In TCP/IP, the transport

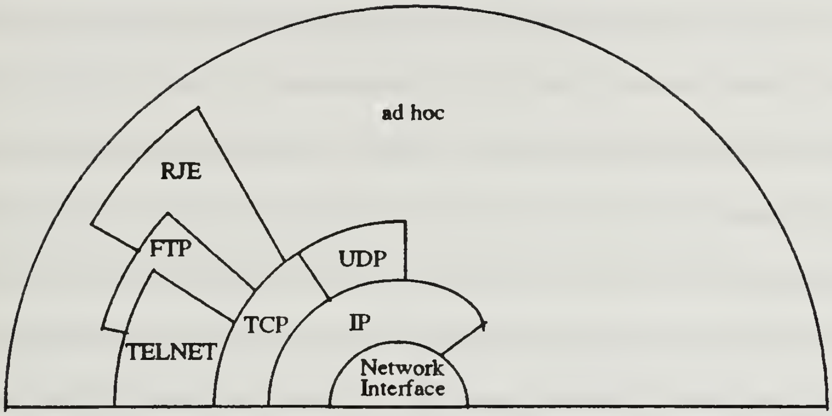


Figure 2 : TCP/IP Communication Architecture Model

protocol is called the Transmission Control Protocol (TCP), which partially "surrounds" the Internet Protocol (IP).

A user of a communications system must have confidence that information intended for transmission to a remote site will get to that site without loss or error. Because the quality of the communication medium may vary drastically from connection to connection, and because the nature of the connection itself may vary from a point-to-point connection to one with multiple intermediary entities, a transport protocol must provide a range of services to meet the needs of the particular application involved. In general, four basic responsibilities are covered: connection management, acknowledgment of received packets, flow control, and error detection and recovery. Though a particular application may or may not require all of these functions, a transport protocol must be capable of providing them.

1. Connection Management.

The service provided by a transport protocol may be either *connectionless* or *connection oriented*. In connectionless mode data packets (datagrams) are routed individually through the network according to routing algorithms which attempt to optimize network performance in terms of point-to-point delay and congestion control. In connection oriented (*virtual circuit)* mode the transport protocol is responsible for establishing, maintaining, and properly closing a connection between two communicating entities. Once this connection has been established, all packets follow the same route between communicating entities and therefore arrive in proper sequential order.

Connectionless service is desirable for short messages which may be confined to a single packet because there is no overhead associated with establishing and maintaining a communication route. Additionally, connectionless mode is more robust in adapting to a dynamic network environment. Whereas a virtual circuit connection would be unexpectedly disconnected if any of the nodes or channels along its route failed, a datagram service could adapt to network failures by rerouting packets along a path that avoided the failed components.

For transfer of larger files which are split up amongst many packets, connectionless mode requires the transport protocol to manage the sequencing of those packets, so that if packets which traveled on different routes are received out of order, they may be properly reordered. Normally, connection oriented mode is used for transfer of voluminous data because the overhead necessary to establish the connection is more than offset by the convenience of sequential data packet delivery.

During connection establishment a number of parameters for the transfer of data must be negotiated. The degree of reliability required is a determining factor in initializing parameters which refine flow control algorithms, error detection and correction capabilities, and acknowledgment schemes. Additionally network parameters such as maximum bandwidth, buffer space availability at the receiver, and round trip delay times must be determined. The transport protocol, either by using pre-established default values or by querying the user or application for desired values, must initialize all communication parameters necessary to manage the transfer of data.

When a connection involves communicating entities which are capable of handling multiplexed channels, a determination must be made as to whether control packets will be interspersed amongst data packets (*in-line)*, or whether they will be transferred separately on a different logical connection (*out-of-line*). The advantage of the out-of-band method is that data packets can be processed in parallel with control packets, avoiding the need for parsing to determine the content of an incoming packet. In some cases, however, it may be advantageous to intersperse control and data packets. In cases where the amount of data to be transferred can be contained in a single packet along with control information, that packet could serve to both establish a connection and transfer the data. Again, it is the nature of the connection that will determine the optimal method for controlling the transfer of data, and it is the transport protocol which must establish the connection environment according to the requirements of the connection.

2. Acknowledgment of Received Packets

Acknowledgment of packets by the receiving entity is often tied into mechanisms for performing flow and error control, though in a strict sense there is a purpose to acknowledgments that is unrelated to either.

The transmitting entity temporarily stores a copy of every data packet which is transmitted to the receiver. When the receiver provides information to the transmitter concerning which packets have been received (*positive acknowledgment)* and, in some cases, which packets have not been received (*negative acknowledgment*), the transmitter is allowed to update its current *state* by discarding from the storage buffer all data packets which have been positively acknowledged. This act, in and of itself, may or may not lead to further actions involved with flow and/or error control. In every case, however, it allows the transmitter to transition from a state which is temporarily out of synchronization with the receiver to a state which is in synchronization with the receiver's state at the time of the acknowledgment. Thus the transmitter and receiver are unified in their view of the *connection state*. At the lowest level, this is the purpose of the acknowledgment process.

In many cases, variables which are used to control the acknowledgment process are *overloaded* for use in both flow and error control mechanisms [CLAR88]. As will be pointed out later, this practice may lead to problems in efficiency where a single lost packet may hold up transmission of other packets.

3. Flow Control

At the transmitter the data to be transferred must be "framed", or inserted into an appropriate packet format, before it can be transmitted. This encapsulation of the data allows each protocol layer which is involved in the process to add its own control information onto the outside of the packet that is passed to it, much as if each layer was putting its own envelope around the data before passing it on to the next layer, and ultimately to the receiving entity. A length field and a checksum for error detection is normally included in each layer's control information to allow the lower layers of the receiving entity's architecture to determine whether the packet arrived error free.

Once a packet arrives at the receiving entity, these encapsulation layers must be analyzed and removed at each successive protocol layer until the original data is available for processing. Additionally, the receiving transport protocol must determine whether the arriving packet is in sequence (for datagram service), and must also place it into a reordering buffer for eventual transfer to the receiving entity. The transport protocol must update the state information maintained by the receiver, and must prepare acknowledgment messages to be returned to the transmitter. Normally, the transport layer at the receiving entity must perform much more processing per packet than its peer transport layer at the transmitter.

It is imperative that the transmitter keep itself abreast of the receiver's buffer status, so that data packets will not arrive at a rate which surpasses the receiver's ability to process. If this situation does occur, the receiver's buffer will overflow and subsequent incoming data packets will be lost. Flow control is the mechanism which forces the transmitter to limit transmission of data packets based upon the current state of the receiver's buffer.

The most common method used to implement flow control is the *window.* A window represents the number of data packets that may be transmitted before pausing for receipt of a *window acknowledgment.* A window size is normally based upon the size of the receiver's buffer space, but may be larger. If the window size is chosen to be larger than the receiver's buffer size, then the flow control algorithm must ensure that the number of outstanding data packets (those which have been transmitted but not yet received), when added to the number of packets currently in the receiver's buffer, does not exceed the total buffer size. The transmitter must take this into account when calculating the number of packets that may be transmitted between each acknowledgment from the receiver.

At one extreme, the transmitter would be forced to delay transmission of each data packet until acknowledgment of the previous packet was received (a window size of one packet). This method of flow control is very inefficient and has led to a phenomenon known as the "silly window syndrome" [DOER90]. On the other hand, exceedingly large windows would result in frequent overflow of the receiver's buffer, necessitating retransmission of large numbers of data packets.

4. Error Detection and Recovery

Errors may occur in two primary forms: loss of data, or corruption of data (bit errors). In either case, the only means of recovering from the error is for the transmitter to retransmit the packet(s) that was not received successfully. The mechanisms used to detect and recover from network errors are vital to reliable and timely communications systems.

Once a data packet has been transmitted, there are only two ways that the transmitter can update its state information based upon the external effects caused by the packet [ZHAN86]. First, an external report may be received. For example, an acknowledgment from the receiver may indicate that the packet was successfully received, and that the receiver's state information was updated appropriately.

Second, the transmitter may use *local detection means*, i.e. timers, to determine that either the packet, or the acknowledgment from the receiver, was lost. If no report is received within a predetermined amount of time, the transmitter will assume that the packet was lost and will retransmit it. Note that in reliable systems a timer must always be used, at least as a backup, to recover from lost acknowledgments. Timer values are based upon an estimate of the round trip delay (RTD) between the two communicating entities. The goal of timer algorithm designers is to minimize the amount of time necessary to detect a loss, while at the same time minimizing false alarms which trigger superfluous retransmission of data packets.

Error recovery mechanisms generally take one of two forms [DOER90]: Positive Acknowledgment with Retransmission (PAR) or Automatic Repeat Request (ARQ). With PAR, the receiver only acknowledges packets which have been received error free and in sequence. Those packets which are not acknowledged will automatically be retransmitted once they have "timed out", or exceeded their timer value. With ARQ, packets are retransmitted based upon specific information contained in an acknowledgment. This applies to window acknowledgment, where the packet with the sequence number below which every other packet has been successfully received will mark the beginning of the window.

One method used to recover from lost packet errors is the *Go-Back-N (GBN)* method. Once a lost packet has been identified, it and all subsequent packets are retransmitted to the receiver. This method must always be used with PAR, and with ARQ if the receiver cannot buffer out of sequence packets. The obvious disadvantage to GBN is that many packets that were successfully received will be retransmitted.

If the receiver has a reordering buffer in which out of sequence packets can be handled, then a *selective retransmission* method may be used with which only the lost packet is retransmitted. This increases the processing requirements of the receiver, but may be desirable in long delay or high bandwidth connections in which a great amount of data may be in transit.

C. DESIGN CONSIDERATIONS

Prior to the introduction of fiber optic transmission media, the limiting factor in throughput was the transmission media itself. That is, the transport protocol at the receiving entity could easily process incoming packets at a rate which exceeded the throughput capacity of the media. Thus, the focus of most attempts to increase throughput was concentrated on minimizing the number of bits required to complete a transfer of data.

Bit packing was often used to "get the most" out of every packet that was transmitted. Additionally, variable length packet fields were used to further minimize the number of bits. Though both of these methods placed an additional processing requirement upon the receiver's transport protocol, the processing rate still exceeded the throughput rate of the medium.

Another problem with pre-fiber transport protocols was packet field limitations. In [LAPO91] the following example is provided:

> A cross continental link between New York and California might have a propagation delay of 30 ms. TCP supplies 16 b for byte-based credit to transmit data. Therefore, TCP can have 65,000 unacknowledged bytes outstanding. At 1 Gb/s this amounts to 0.5 ms of transmission. After sending this data, the transmitter must sit idle for approximately 60 ms waiting to get additional credits to transmit data from the receiver. Obviously, this yields unacceptable throughput performance. If the credit field were enlarged to 24 b, the transmitter could send for 134 ms, or about 134 Mb of data before exhausting its credit. This would be sufficient time to receive additional credit from the receiver.

Flow control mechanisms were often designed as a safeguard against exceptional situations rather than as a function of primary importance. The consideration was not whether the receiver could "keep up" with the transmitter's transmission rate, but whether the transmission rate was saturating the communication channel; a situation marked by an increase in the number of lost packets.

Network error rates were also much more common with cable than with fiber-optic media. Error rates of 1 in 10^6 were common, forcing protocol designers to include very complex error detection and recovery mechanisms. Designing protocols to compensate for failure was a primary practice that necessitated the design of very robust, bulky protocol standards. As will be discussed in a later section of this thesis, the low error rate of fiber (1 in 10^9) allowed designers to adopt a new paradigm of designing for success, rather than preparing for failure.

D. PROBLEMS WITH EXISTING TRANSPORT PROTOCOLS

The introduction of fiber optic media proved to be the catalyst for what has become a communications revolution. The bandwidth available to transfer information has allowed the dreams of just ten years ago to become reality. As applications which require the quality of service provided by fiber optics have been developed, however, it is apparent that existing transport protocols will not suffice. In this section, several problems which are hindering utilization of the full potential offered by fiber optics will be discussed.

1. Timers and Round Trip Delay Estimation

As discussed in section II.B.4 (page 11), timers are both a necessity and a weak point of transport protocols. The purpose of timers is to alert the transport protocol of a possible lost packet or connection failure. In [ZHAN86], the author explicitly details the poor performance of timers in the TCP standard. It is important to note that it is the inherent nature of timers that cause problems rather than the specific implementation of timers in the TCP protocol. Because timers are alarms which are based upon incomplete information, the recovery technique used to handle them must be non-optimal.

State synchronization between two communicating entities is based upon the exchange of control information. When that control information is not available to one of the entities, there exists a situation in which an assumption must be made as to the state of the other. The method used by timing mechanisms to determine when a failure might have occurred is to estimate the round trip delay (RTD), and to take action when an expected response has not been received within that time period. The problem is in determining what value to use for the RTD period.

In a datagram environment such as the Internet, the route taken by any two data packets may be different, depending on congestion or network failures. Furthermore, even if the route is the same, delays at either end entity, or any intermediate entity, may also vary according to the current state of the network. The overall variation in RTD can be quite significant, and therefore any estimate of the RTD must be assumed to be questionable at best.

If a timeout value is chosen which is too small, then many packets which may have only been delayed will be assumed to have been lost, leading to superfluous retransmissions which will add to network congestion and compound the problem. Additionally, reception of duplicate packets will increase the processing burden at the receiver, also adding to the time that it will take for an acknowledgment to be received by the transmitter, and again triggering additional unnecessary retransmissions.

A timeout value which is excessive will cause unnecessary waiting for valid retransmissions by the transmitter, thus decreasing performance of the system and negating the advantages made available by fiber optic technology. In [ZHAN86], a case is documented in which retransmission delays went from several hundred milliseconds to over two minutes. This is obviously unacceptable.

Some algorithms have been developed which modify the timeout value dynamically, based upon the current *perceived* state of the network. These algorithms provide somewhat better performance, but still lead to retransmission surges at the beginning of a communication session, as there is no way to predict the initial RTD between two entities.

As discussed earlier, timers are a necessary backup to explicit failure reporting systems because those systems themselves may fail. Systems which rely upon timers as their primary failure detection mechanism will continue to exhibit poor performance in the fiber optic age. In the next chapter, some alternative mechanisms will be discussed which attempt to circumvent the inherent weakness of timers.

2. Go Back N Method of Error Recovery

When the amount of data which is enroute between the transmitter and the receiver is relatively small, the Go Back N (GBN) method may prove to be adequate for recovering lost packets. Indeed, in some systems GBN must be used (page 13). However, as alluded to in the introduction of this thesis, in long haul fiber optic networks, up to 30

Mbits or more of data may be outstanding at any given time. Doubling that amount to account for RTD would create a situation in which 60 Mbits of data would have to be retransmitted to recover a lost packet using GBN.

If the receiver is capable of buffering out-of-sequence data packets, then a selective retransmission scheme may be implemented in which only the lost packet is sent again. This places an increasing processing requirement on the receiver, however, and may still require timers to determine when a particular packet may be assumed lost.

While it is generally accepted that some form of selective retransmission must be implemented in fiber optic networks, there exist a variety of research approaches which attempt to optimize performance for error recovery. Some of these techniques will be discussed in the next chapter.

3. Flow Control Tied to Error Detection and Recovery

The use of fiber optic media in communications networks has greatly increased the need for high performance flow control algorithms. Whereas in existing transport protocols flow control mechanisms are commonly tied in with error detection and recovery, optimal high speed network performance will require a separation of the two functions. The following example illustrates the problem [CLAR88]:

> Assume that a protocol uses a window acknowledgment scheme with a window size of *10*. Assume also that the following situation exists: the transmitter has transmitted *12* packets of data, and packet number *2* is lost; the receiver has sent an acknowledgment to the transmitter indicating that packet number *2* has not yet been received.
>
> Because the transmitter has already transmitted the next *10* packets after *2*, (the protocol window size), it is prevented from transmitting any additional packets until the *2nd* packet is retransmitted and received error free by the receiver, and the acknowledgment indicating the increased window allotment has been received by the transmitter. This evolution takes *at least* one RTD, and thus causes throughput to be controlled by how quickly the recovery can be performed.

Future flow control mechanisms will have to address *how fast* packets may be transmitted, rather than *how many* may be outstanding at a given time. The practice of overloading variables to control both flow and error recovery will have to be abandoned. In the next chapter a new method of *rate control* will be discussed in which packets are transmitted according to a pre-negotiated *number per time period.*

4. Non-Standard Formats

Because one of the primary design features of past transport protocols was bit-packed packet architectures, variable length and variable format fields were often used to minimize the size of data and control packets. The amount of processing required to decode the packets at the receiving entity was not a limiting factor in terms of throughput because the packet processing rate exceeded the packet arrival rate.

With the order of magnitude increase in raw bandwidth made available by fiber optic media, this practice is no longer acceptable. The most significant performance measure of any future transport protocol will be its ability to avoid buffer overflow by passing incoming packets on to the receiving host with a minimal amount of processing. To achieve this goal, designers are developing operations to quickly interpret and route the various components of *standardized* packets to their appropriate place within the receiving entity's architecture.

To further enhance performance, many protocols will use multiple processors to process the various components of the data packet in parallel. By using standard format packets, with fields divided logically in lengths of *computer words*, the parallel processing method will significantly reduce the time necessary to process a data packet.

Another approach being taken is to embed as much of the protocol as possible into the hardware of the receiving entity's protocol processor. For systems which are consistently used to communicate with standard and constant *connection environments*, specialized hardware architectures will be developed to completely optimize transport protocols.

With both of these approaches, an increase in bandwidth necessary to achieve standardization is acceptable. Because systems of the near future will not come close to using the full available bandwidth offered by fiber optics, the inefficiencies caused by variable packet architectures will prove negligible.

III. SURVEY OF LIGHTWEIGHT TRANSPORT PROTOCOLS

Currently published papers in the field of lightweight transport protocol design consistently refer to four primary research projects which offer different solutions to the problems discussed in the previous chapter. The purpose of this chapter is to discuss some of the mechanisms which are being used to achieve an increase in protocol performance. The protocols which will be discussed are: Network Bulk Transfer (NETBLT) (Massachusetts Institute of Technology), Versatile Message Transaction Protocol (VMTP) (Stanford University), and the Xpress Transfer Protocol (XTP) (Protocol Engines Inc.).

A. NETWORK BULK TRANSFER

The major goal of Network Bulk Transfer (NETBLT) is high throughput. In order to use as much of the available raw bandwidth as possible, the designers chose to implement *rate control*, in which the transmitter sends a predetermined number of packets per time interval dependent upon the current network congestion and the receiver's ability to process incoming packets. This rate is independent of round trip delay (RTD) and is separate from the error recovery mechanism of the protocol.

Using rate control, a minimum time period for packet transmission is established. This could lead to problems with overhead if timing mechanisms were required to work at the granularity of a single packet. To alleviate this condition, the transmission is controlled in groups of packets, called *buffers*, and is measured in terms of a burst size (the number of packets in a burst) and a burst rate (the number of milliseconds between the start of one burst transmission and the start of the next). Using these parameters, the protocol is able to more accurately reflect the current state of the receiver's resources. For example, they could represent a slow machine with little buffer space (long burst interval, small burst size), or a faster machine with a high process-scheduling overhead (short burst interval, large burst size) [CLAR88].

Error recovery is also performed at the buffer level. According to the protocol design, no packets from a buffer will be sent before all packets from the previous buffer have been sent. The transmitter begins sending data packets at the predetermined rate upon synchronization with the receiver (accomplished through a transmission message). The receiver starts a timer upon receipt of the first packet, and by knowing the transmission rate, calculates the time expected to receive the entire buffer. At the end of this time, if all packets from the buffer have not arrived, the receiver will send a list of all missing packets to the transmitter. The transmitter will then retransmit the missing packets in the same burst with new data packets from the next buffer. Because the packets are identified by buffer, the receiver does not mix them up. If all packets from a buffer are received, the receiver sends an acknowledgment of the buffer back to the transmitter.

Testing of the NETBLT protocol resulted in the determination that initial rate control values must be based upon information available from lower network layers. The current state of congestion in the network is known most precisely be the gateways which service the connection. More work is being planned to develop methods of obtaining congestion information from these gateways.

B. VERSATILE MESSAGE TRANSACTION PROTOCOL

Versatile Message Transaction Protocol (VMTP) also uses selective retransmission to accomplish error recovery. In contrast with NETBLT, the designers of VMTP state that many hosts find it easier and more efficient in processing cost to send a large amount of data as a single blast of packets, transmitted as fast as the network allows, rather than pacing the flow [CHER86]. So, while VMTP also uses the mechanism of grouping packets for transmission, it does not use rate control in the same manner as NETBLT to monitor the transmission of these groups.

The goal of VMTP is to allow the transmitter to recognize the occurrence of a situation in which the transmission rate is consistently overflowing the receiver's buffer. By analyzing a *bit mask* from the receiver indicating which packets are missing, the transmitter determines whether a pattern exists in which every k^{th} packet, for example, may be lost. If a cyclic pattern of loss exists, the transmitter assumes that it is transmitting at a frequency which is a constant value greater that the receiver's processing rate, and adjusts its transmission rate accordingly. The difference between this method and NETBLT's flow control is that there is no synchronization process between the sender and receiver, and the receiver has no active role in determining the transmission rate.

Flow control in VMTP is accomplished in units of *packet groups*. In the server-client environment for which VMTP was specifically designed, the client must be prepared to accept a full packet group before any packet from the group may be transmitted. In this way, the server's resources are not tied up waiting for the client to authorize the sending of additional packets, as with a sliding window protocol. This is a significant achievement for a system in which many clients may be simultaneously requesting data transfers from a single server.

C. XPRESS TRANSFER PROTOCOL

Unlike previous transport layer protocols, Xpress Transfer Protocol (XTP) is being designed to be implemented in hardware as a VLSI chip set. Additionally, XTP will combine the transport and network layers to achieve an integrated capability to dynamically adjust transmission parameters based upon current network states.

Flow control in XTP is achieved through the use of parameters which provide visibility of the receiver's buffer to the transmitter. These parameters indicate which packets have been passed to the receiving host, which are currently in the buffer, which have not been received (gaps in the monotonically increasing sequence), and the current buffer space available to accept new packets. The parameters are based on units of bytes, rather than packets, to increase the efficiency with which buffer management processing may be accomplished. Using this method of flow control, the transmitter bases transmission of additional packets upon the current state of the receiver's buffer, rather than on the existence of missing packets. In other words, flow control is effectively separated from error reporting.

In addition to flow control parameters, XTP uses rate control to adjust the transmission of data packets. Although the flow control/buffer visibility scheme discussed above provides the transmitter with some insight regarding the receiver's resources, it does not provide any information regarding the receiver's ability to manage the buffer. Rate control, as with the NETBLT protocol, guarantees that the receiver will have enough time to process back-to-back packet bursts before the next burst is transmitted. Rate control and flow control together allow XTP to fine tune the transmission of data packets to a level that is optimal, based upon the current state of the network and the receiving host. Furthermore, by encompassing the network layer functions, the XTP transmitter gains additional insight into the state of the network to determine initial burst rates for data packets.

Three methods of error recovery are possible with XTP. First, once the receiver receives an out of order sequence number, it can determine that a gap exists and that packets may be lost. It can then inform the transmitter of this fact and await retransmission.

Alternatively, after transmitting a predetermined group of bytes, the transmitter may request from the receiver a list of gaps for which sequence numbers are missing. The receiver sends the gap list to the transmitter, and selective retransmission is performed. If the transmitter does not receive any indication from the receiver, it will eventually timeout, thus prompting an additional request for gaps.

D. SUMMARY

The common threads behind all of these protocols are the grouping of packets into larger entities for purposes of error recovery and flow control, and the inclusion of some form of rate control for dynamically adjusting the number of packets which are transmitted during a given time period. Selective repeat is the common error recovery method, as the go-back-N method would require retransmission of too much properly received data. As will be demonstrated in the next chapters, the separation of flow control and error recovery, which is incorporated into all of the protocols discussed in this chapter, is a necessary design criteria for future transport protocols.

IV. THE AT&T TRANSPORT PROTOCOL

A. DESIGN PHILOSOPHY

In any reliable communication system there must exist a mechanism for synchronizing the states of the communicating entities. The purpose of this state synchronization process is to provide visibility of the system *as a whole* to each of the individual components of the system. Using this information, the component entities may adjust their execution, if necessary, to bring the system to a stable and steady state. Without this mechanism, each of the component entities would be isolated, and the communication system would be reduced to an unreliable "shoot and run" process in which the fate of a transmitted message would be unknown.

In transport protocols it is the function of control packets to synchronize the states of the communicating entities. The exchange of state information allows flow control and error recovery to be performed, thus providing a degree of reliability that is necessary for most communication processes. Normally, control packets are transmitted based upon the occurrence of some significant event, such as the receipt of a data packet or timeout of a retransmission timer.

In the AT&T Transport Protocol a different approach is taken. The designers chose to exchange complete state information between the communicating entities on a periodic and frequent basis, independent of significant events which may have occurred. This method reduces the complexity of protocol processing by removing many of the procedures required to recover from network inadequacies such as bit-errors, packet loss, and out of sequence packets, and makes it more amenable to parallel processing [NETR90]. Additionally, the protocol uses a modified selective repeat error recovery mechanism to perform packet retransmission efficiently.

B. ORGANIZATION

The protocol model consists of eight finite state machines (FSM); four for the transmitting entity (labeled T1 through T4) and four for the receiving entity (R1 through R4). Each of the four machines at a given entity carries out a specific function and executes independently, thus increasing the potential for a parallel processing implementation of the protocol. The general organization of the machines is illustrated in Figure 3. The arrows in the figure represent communication from one machine to another.

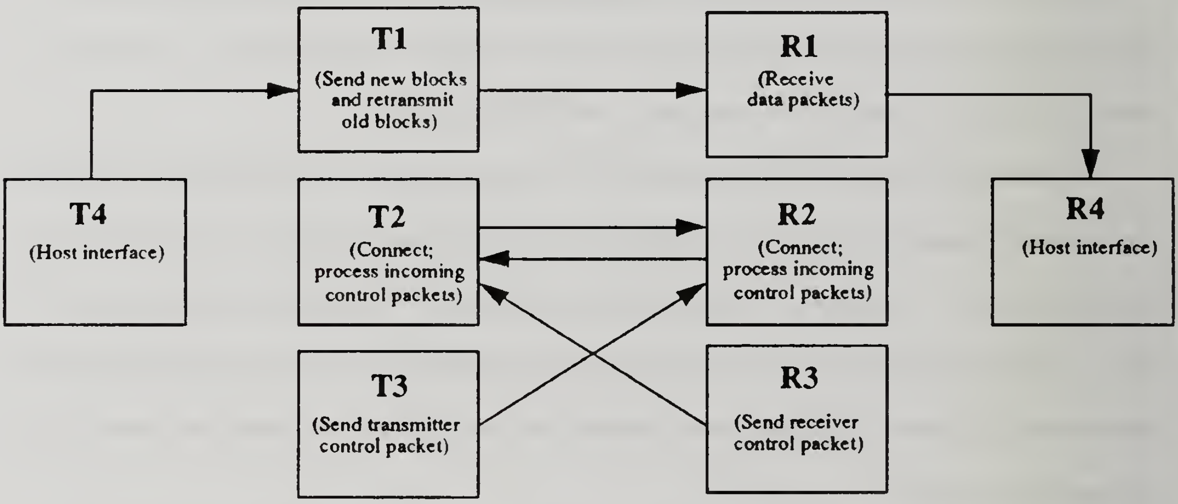

T1
(Send new blocks and retransmit old blocks)
R1
(Receive data packets)
T4
(Host interface)
T2
(Connect; process incoming control packets)
R2
(Connect; process incoming control packets)
R4
(Host interface)
T3
(Send transmitter control packet)
R3
(Send receiver control packet)

Figure 3 : Machine Organization

C. MODES OF OPERATION

The protocol provides a degree of flexibility by allowing selection of three modes of operation. In the first mode (mode 0) no flow control or error recovery is executed. This may be appropriate in systems which have frequently transmitted, short life data packets. In a remote sensing application, for example, environmental data may be transmitted many times per second, with each transmission making the value from the previous transmission obsolete.

Mode 1 provides flow control, but no error detection or recovery. Mode 2 is the most reliable mode, providing both flow control and error recovery. The specification in this thesis does not allow dynamic modification of the operating mode, though theoretically there is no reason why alteration of the mode could not be implemented.

D. PACKETS AND BLOCKS

To decrease the amount of congestion in the network, the number of acknowledgments transmitted by the receiver is minimized by acknowledging packets in groups called *blocks*. When a block acknowledgment is received by the transmitter, all packets within that block are effectively acknowledged. Thus, if there are eight packets per block, the transmission requirement upon the receiver is decreased eight times.

Similarly, if a single packet from a block is lost, all packets from that block are retransmitted. The careful reader may wonder whether the increase in retransmitted blocks will outweigh the savings in packet acknowledgments. The point to remember is that lightweight protocols are designed for *success*, therefore over a period of time, the savings in packet acknowledgment will be more significant.

E. CONNECTION ESTABLISHMENT

Establishment of a connection involves the standard three-way handshake, and includes negotiation of certain parameters which will effect the execution of the protocol. The negotiated parameters include: mode, bandwidth, packet size, block size (in units of packets), and buffer size. In stable network environments, default values for these parameters may be relied upon to decrease the burden of negotiation. Additionally, it is assumed that an estimate of the round trip delay can be calculated during the connection establishment process. This value will be used to initialize certain local variables in machines T3 and R3, and to determine a value for the necessary buffer size at the receiver.

F. BUFFERS

A buffer is required at the receiver to provide a capability to temporarily hold and/or resequence data packets prior to their being passed on to the receiving host. In order to prevent frequent overflow of this *reordering buffer*, it must be large enough to hold the equivalent of the amount of data that may be transmitted during any RTD period. The reason for this is that it takes *at least* one RTD for the transmitter to be notified that a packet has been lost. During this time, the transmitter will continue to transmit new data packets, which the receiver may not pass on to the host until the earlier lost packet is received. Remember that one of the purposes of a transport protocol is to deliver data packets *in sequence* to the receiving host. The size of the reordering buffer for this protocol is calculated as *RTD* x *bandwidth.*

G. NOTATION

The following notation is defined in [NETR90], and is presented here for continuity.

- LW_t (LW_r) is the maximum sequence number of the block below which every packet in every block has been correctly received as known at the transmitter (receiver).

- UW_t (UW_r) is the maximum sequence number of the block below which every block has been transmitted but not necessarily acknowledged as known at the transmitter (receiver).

- LCI is the logical connection identifier assigned to each logical connection during connection-establishment phase. Each logical connection has its own pre-negotiated buffer at the receiver.

- L is the largest allowed number of outstanding blocks chosen at the connection establishment. L is chosen to be slightly larger than

  [(*RTD* x *peak bandwidth*) / *number of data bits in a block*] and can be thought of as a window.

- LOB is a bit map representing the outstanding blocks between LW_r and (LWr+L-1).

- NOU (<L) is the number of outstanding blocks which have been transmitted but not yet acknowledged by the receiver.

H. THE EVENT *clock-tick*

The predicate *clock-tick* is a periodic event which occurs in intervals of time determined by the round trip delay and data packet transmission rate. This event is used to initiate an evaluation of internal state variables to determine whether some action should be taken, such as transmission of state information, retransmission of a data packet or connection message, etc. It is helpful to think of *clock_tick* as a timing mechanism for the protocol.

V. SYSTEM OF COMMUNICATING MACHINES (SCM)

In this section the model used to specify and analyze the protocol is described. A more detailed description appears in [LUND88].

A *system of communicating machines* is an ordered pair $C = (M,V)$, where

$$M = \{m_1,m_2,...,m_n\}$$

is a finite set of *machines*, and

$$V = \{v_1,v_2,...,v_k\}$$

is a finite set of *shared variables,* with two designated subsets R_i and W_i specified for each machine m_i. The subset R_i of V is called the set of *read access variables* for machine m_i, and the subset W_i the set of *write access variables* for m_i.

Each machine $m_i \varepsilon M$ is defined by a tuple (S_i,s_0,L_i,N_i,τ_i), where

(1) S_i is a finite set of states;

(2) $s_0 \varepsilon S_i$ is a designated state called the *initial state* of m_i;

(3) L_i is a finite set of *local variables*;

(4) N_i is a finite set of names, each of which is associated with a unique pair (p,a), where p is a predicate on the variables of $Li \cup Ri$ and a is an *action* on the variables of $Li \cup Ri \cup Wi$. Specifically, an action is a partial function

$$a{:}Li \times Ri \rightarrow Li \times Wi$$

from the values contained in the local variables and read access variables to the values of the local variables and write access variables.

(5) τ_i: $Si \times Ni \rightarrow Si$ is a transition function, which is a partial function from the states and names of m_i to the states of m_i.

Machines model the entities, which in a protocol system are processes and channels. The shared variables are the means of communication between the machines. Intuitively, R_i and W_i are the subsets of V to which m_i has read and write access, respectively. A machine is allowed to make a transition from one state to another when the predicate associated with the name for that transition is true. Upon taking the transition, the action associated with that name is executed. The action changes the values of local and/or shared variables, thus allowing other predicates to become true.

The set L_i of local variables specifies a name and a range for each. The range must be a finite or countable set of values.

A *system state tuple* is a tuple of all machine states. That is, if (M,V) is a system of n communicating machines, and s_i, for 1<i<n, is the state of machine m_i, then the n-tuple $(s_1, s_2, \ldots, s_n)$ is the system state tuple of (M,V).

A *system state* is a system state tuple, plus the outgoing transitions which are enabled. That is, two system states are *equivalent* if every machine is in the same state, *and* the same outgoing transitions are enabled.

The *initial system state* is the system state such that every machine is in its initial state, and the outgoing transitions are the same as in the initial global state.

The *global state* of a system consists of the system state, plus the values of all variables, both local and shared. It may be written as a larger tuple, combining the system state with the values of the variables. The *initial global state* is the initial system state, with the additional requirement that all variables have their initial values. A global sate *corresponds* to a system state if every machine is in the same state and the same outgoing transitions are enabled. That is, a global state consists of a tuple of machine states, plus the values of all variables. A system state with the same tuple of machine states and the same enabled outgoing transitions is the corresponding system state.

Let $\tau(s_1,\mathrm{n}) = s_2$ be a transition which is defined on machine m_i. Transition τ is *enabled* if the enabling predicate p, associated with name n, is true. Transition τ may be executed whenever m_i is in state s_1 and the predicate p is true (enabled). The *execution* of τ is an atomic action, in which both the state change and the action *a* associated with *n* occur simultaneously.

Note that if the values of all variables are restricted to some finite range, then the model can theoretically be reduced to a simple finite state machine. Otherwise, an infinite number of global states are possible. However, even if the number of global sates is infinite, the number of system states is finite, because of the finiteness of each machine. This may allow a reachability analysis on the system states, when a reachability analysis on the global states is infinite. Even when the values of all variables are of a finite range, the number of global states in the equivalent FSM system may be so large as to be intractable. In this paper, it is shown how this model can reduce these difficulties for a specific protocol.

VI. SPECIFICATION STRUCTURES

A. PURPOSE AND SCOPE

The purpose of a formal specification is to clearly define a protocol's behavior in all possible states, and to provide a vehicle which may be used to demonstrate its execution. The SCM model uses a combination of FSM's and *Predicate Action Tables* (PAT) to provide a framework through which full or partial analysis may be performed. The PAT contains enabling predicates and resulting actions for every transition in the system, while the FSM's provide a view of the individual machine states as well as the overall system state. Beginning in the initial system state, the FSM's provide the set of all possible outgoing transitions, and the PAT reduces the set of transitions to those which are currently enabled. In order to follow the specification for this protocol, it is necessary to be familiar with the various structures and variables which are referenced in the *enabling predicate* and *action* sections of the PAT.

B. COMMUNICATION STRUCTURES

Each of the machines in the protocol performs a specific function in coordination with one other machine according to the organization presented in Figure 3. The machines communicate through the use of *shared variables* which affect the truth or false state of the enabling predicates, and through *communication channels* which carry messages from one communicating entity to the other. The communication channels are modeled as queues with message reordering and deletion allowed. The reordering and deletion is performed by an implicit channel machine (demon) which has only those two operations. The following communication structures are used to effect the transfer of data and control packets, and to maintain state information used for flow control and error recovery.

1. T_CHAN

T_CHAN is the communication channel from the transmitter to the receiver. It holds message elements which represent data packets, transmitter control packets, connection requests, connection acknowledgment confirmations, and disconnect messages. All of these message elements are considered to be of the same type for this specification. Messages dealing with connection management are enqueued into *T_CHAN* by machine T2 and dequeued by machine R2 during the connection establishment phase and at the close of the connection. During the data transfer phase, machine T1 enqueues data packets, while machine T3 enqueues transmitter control packets. Machine R1 removes data packets from *T_CHAN*, and machine R2 removes transmitter control packets.

2. R_CHAN

As the mirror of *T_CHAN, R_CHAN* handles all message elements transmitted from the receiver to the transmitter. The only message elements transmitted by the receiver are connection acknowledgments and receiver control packets, both of which are removed by machine T2.

3. OUTBUF

OUTBUF is a buffer space into which machine T4 deposits all data packets for transmission to the receiver. Rather than having the transmitting host be tied up by network delays, it can dump as much data as possible into the buffer space and divert its processing time to other applications as it awaits notification that the transfer has been completed, or that the buffer is empty. Machine T1 removes data from *OUTBUF* and inserts it into data packets for transmission to the receiver.

4. INBUF

The purpose of *INBUF* is to temporarily hold incoming data packets for one of two reasons: either the receiving host is not ready to receive new packets, or the data packets have arrived out of sequence and must be reordered and/or held pending retransmission of a lost packet. The size of *INBUF* is critical for proper flow control, and must be able to hold at least an amount of data equal to the *bandwidth delay product* (*RTD* x *bandwidth*). At the host's convenience, data packets will be removed from *INBUF* from machine R4 as long they are in proper sequential order. Data packets are placed into *INBUF* by machine R1 upon retrieval from *T_CHAN*.

5. LUP TABLE

The LUP is the structure through which error recovery is performed. Each time a data packet is transmitted, a copy of the data field from the packet is placed into the LUP along with a *count* value for the block which the packet was contained in. The *count* field of each LUP record is initially set to (RTD / T_{in}) + *cons*, where *cons* is a constant (e.g. 2) and T_{in} = max (RTD / *kou* , *IPT*). The constant *kou* is typically a power of 2, such as 32, and *IPT* is the average time between two data packet transmissions.

Thereafter, *count* is decremented by *k* every time the transmitter receives a control packet from the receiver, where *k* is the interval between two control packet transmissions expressed in units of T_{in}. The value of *k* is initially set equal to T_{in}, but is doubled each time that the logical connection is inactive since the last control packet transmission, up to a maximum value of max (RTD / *m* , *IPT*), where *m* is another constant (e.g. 8). Upon transmission of the next data packet, *k* is reset to T_{in}.

The effect of this somewhat complex formula for determining the retransmission timeout value (*count*) is that the protocol will automatically adjust itself based upon the amount of data traffic being transmitted. Thus, the more data that is transmitted, the lower the initial value of *count* will be. The result of this dynamic adjustment process is that the throughput of the protocol is almost independent of the value of the round trip delay [NETR90].

A block is scheduled for retransmission when its *count* value reaches zero prior to acknowledgment. Upon retransmission of a block, all of the LUP entries containing packets from that block have their *count* value reset. When a block is acknowledged, all of the records containing packets within that block are removed from the LUP. The advantage of maintaining the LUP table in this manner is that explicit timers are not required to perform error recovery.

6. *AREC* AND *RECEIVE*

AREC and *RECEIVE* are tables maintained by the receiver which are updated as data packets are received. Initially, all elements of both tables are set to 0. *RECEIVE[i]* is set to 1 upon receipt of the i^{th} data packet of a block, and *AREC[j]* is set to 1 upon receipt of all packets from the j^{th} block. *AREC* and *RECEIVE* provide the capability for selective repeat and allow increased throughput by managing multiple blocks of data packets simultaneously.

The *LOB* bit map uses *AREC* and *RECEIVE* to provide acknowledgment information to the transmitter through the receiver control packet.

C. MESSAGE TYPES

There are no explicit type declarations within the specification. For the purposes of this model, messages can be thought to belong to one of four categories: connection messages, receiver control packets, transmitter control packets, and data. The following paragraphs describe these categories in more detail.

1. Connection Messages

Connection messages can be one of four types: connection request (*Conn_Req*), connection acknowledgment (*Conn_Ack*), connection confirmation (*Conn_Conf*), and disconnect (*Disc*). The connection establishment process follows the standard three-way handshake.

Upon receiving a signal from its host, the transmitter sends a request for connection, which contains recommended values for negotiated parameters, to the receiver. The receiver analyzes the request, and if satisfied with the recommended values simply returns an acknowledgment of the request. If the receiver cannot operate under the parameters recommended by the transmitter, it will include modified values for those parameters in its acknowledgment. The transmitter must then analyze the returned acknowledgment to determine whether to accept the connection, in which case it sends a connection confirmation to the receiver, or to reject the connection, in which case it will send a disconnect message to the receiver and notify the host of the failed connection attempt. Some implementations may allow the three-way handshake to begin again automatically upon determination of a failed negotiation, repeating the process until parameter values are eventually agreed upon.

In the event that a connection confirmation message from the transmitter to the receiver is lost, and the receiver receives the first data packet when it expects the confirmation, the connection will be implicit and the receiver will accept the data.

2. Receiver Control Packets

Receiver control packets contain the state information necessary for the transmitter to determine which actions are required to become synchronized with the receiver. The format of the packet is shown in Figure 4.

LCI	Type=0	Seq #	k	LW_r	Buffer_available	LOB	Error Check

Figure 4 : Receiver Control Packet Format

The variable used to hold the instantaneous values of the receiver's state is *R_state*. Each time the event *clock_tick* occurs, the receiver will determine whether to send a control packet. This determination is based upon the activity on the logical connection since the last control packet transmission, as discussed on page 35 (LUP TABLE). If a control packet is to be transmitted, it is placed into the *R_CHAN* communication channel (queue) for retrieval by the transmitter.

The *Logical Connection Identifier (*LCI) is used in support of multiplexed connections to a single protocol processor or front end machine. The *type* field identifies the packet as a receiver control packet, and may be used to quickly route packets in parallel processing implementations.

The sequence numbers are maintained separately from sequence numbers of other message types, such as data packets and transmitter control packets, and allow the transmitter to determine whether the packet has arrived in order. Monotonically increasing packets are kept and processed, while out of sequence packets are discarded due to the outdated nature of their variable's values.

The interval between two control packet transmissions is contained in the variable *k*, which is used to decrement the *count* value of LUP records. *Buffer_available* indicates to the transmitter the number of spaces available in the reordering buffer, in units of blocks. This variable is used to control the flow of data packets to the receiver to prevent buffer overflow and subsequent lcss of packets.

3. Transmitter Control Packets

The information contained in the transmitter control packets allows the receiver to be "smart" in the sense that it has some indication of the state of the transmitter. It is sufficient that the receiver play a completely passive role, acknowledging blocks of data as they arrive, and waiting for retransmission of lost packets at the convenience of the transmitter. There is a lot the receiver can do, however, to make the transfer of data from multiple logical connections more efficient, if it has some degree of visibility of the transmitter's state.

The format of the transmitter control packet is shown in Figure 5. The purpose of the first four fields is the same as for the receiver control packet. The *No. of blocks queued* field could be used by the receiver to determine how near to completion the transfer on a particular logical connection is. The receiver could use this information in allocating shared resources amongst several logical connections.

LCI	Type=1	Seq #	k	UW_t	No. of blocks queued	Error Check

Figure 5 : Transmitter Control Packet Format

4. Data Packets

The format of a data packet is shown in Figure 6. Again, the purpose of the first three fields is the same as for the receiver and transmitter control packets.

LCI	Type=2	Seq #	Data	Error Check

Figure 6 : Data Packet Format

D. OPERATIONS

To allow a more abstract modelling of the protocol within the formal specification certain operations (subroutines) will be defined here and referenced from the Predicate Action Table. Each operation will be applicable to particular structures or types previously defined.

1. Operation to Reconcile Outstanding Blocks

The purpose of this operation is to update the variable NOU, which represents the number of outstanding blocks as known by the transmitter. This is accomplished by reconciling consecutive LOB bit-maps, the first of which is maintained by the transmitter in the bit-map variable HOLD and represents the LOB field from the last receiver state packet, and the second of which is the LOB field contained in the current receiver state packet.

```
Operation **Balance**
   Input Parameters: $LW_r$ , $LW_t$, L, HOLD, LOB, NOU
   Output Parameters: NOU

Begin
   for i = $LW_r$ to $LW_r$-1 loop
      if not set(HOLD(i-$LW_t$+1)) then
         dec(NOU);     -- decrement NOU
      end if;
   end loop;

   for i = $LW_r$ to $LW_t$+L-1 loop
      if set(LOB(i-$LW_r$-1)) and not set(HOLD(i-$LW_t$+1)) then
```

```
            dec(NOU);
         end if;
      end loop;

      for i = ($LW_t$+L-$LW_r$+1) to L loop
         if set(LOB(i)) then
            dec(NOU);
         end if;
      end loop;
   End
```

2. Operation to Remove Acknowledged Blocks

The purpose of this operation is to update the LUP table by removing all data packets in blocks which have been acknowledged by the receiver. The LOB bit-map and the variable LW_r, both from the current receiver state packet, are used to accomplish this task.

```
   Operation **Update_LUP**
      Input Parameters: LOB,   LUP,   $LW_r$
      OutPut Parameters: LUP

   Begin
      for all LUP(i) loop
         if LUP(i).seq < $LW_r$ then
            remove(LUP(i));
         end if;
      end loop;

      for all LOB(i) loop
         if set(LOB(i)) then
            remove(LUP(j)) *where* LUP(j).seq = $LW_r$+i-1;
         end if;
      end loop;
   End
```

3. Operation to Insert Received Data into Reordering Buffer

The obvious purpose of this operation is to place the data fields from incoming data packets into a reordering buffer for sequential transfer to the receiving host. The size of the reordering buffer is assumed to be a multiple of the number of packets contained in a block. That being the case, the reordering buffer can work like a *ring buffer* in that modulo arithmetic can be used to "cycle" through the buffer. It is also assumed that L, the limit on the number of outstanding blocks, will be of such a value as to prevent an unchecked completion of an entire cycle through the buffer, and thus an inadvertent overwriting of data. That is, if the receiving host is not retrieving data from the buffer as fast as it is being deposited, the parameter L will cause the transmitter to delay sending data blocks until the receiving host has "caught up".

```
 Operation **Order_Insert**
  Input Parameters: INBUF
                    Pkt (a data packet),
                    i  (the number of packets in a block),
                    buffer_size (in units of blocks)
  Output Parameters: INBUF

Begin
  INBUF((Pkt.seq-1 mod (buffer_size * i)) + 1) := Pkt.data;
  if Pkt.seq-1 mod i = 0 then
     dec(buffer_available);
  end if;
End
```

4. Operation to Retrieve Ordered Data Stream from Reordering Buffer

The receiving host will initiate and accomplish retrieval of data from the reordering buffer. Essentially, as long as there is a steady stream of sequentially ordered data in the buffer, the receiving host will continue to retrieve it, barring some system problem of it's own.

```
Operation **Ordered_Retrieval**
  Input Parameters: INBUF, buffer_size (in units of blocks),
                    i (the number of packets in a block)
  Output Parameters: INBUF
```

```
Begin
   index := 1;     -- some form of indexing variable
   if not empty(INBUF(index)) then
      pass the data stream to the host and clear INBUF(index);
      inc(index);     -- increment index
      if index = buffer_size then
         index := 1;
      end if;
      if index-1 mod i = 0 then
         inc(buffer_available);
      end if;
   end if;
End
```

5. Operation to Acknowledge Receipt of Data Blocks

The *RECEIVE* and *AREC* structures provide the necessary capability to receive and pass data to the receiving host in terms of packets, while acknowledging receipt of data to the transmitter in terms of blocks. As with the operations which manipulate the INBUF structure, this operation will use modulo arithmetic to maintain a "floating window" bit-map representing the outstanding packets and blocks. To reduce the complexity of the operation, a sequential ordering of packet sequence numbers and block numbers will be maintained within the bit-map "window". Thus, if all packets of the i^{th} block are received, but the i^{th} block is not the first block of the RECEIVE bit-map, then the i^{th} block will not be cleared until all blocks below i are cleared. This does not affect the performance of the protocol since L, the limit on the number of outstanding blocks, will drive the number and range of the blocks represented in the RECEIVE structure.

```
Operation Process_Packet
   Input Parameters: RECEIVE, AREC, $LW_r$, $UW_r$, L,
                       X (the sequence number of a data packet),
                       i  (the number of packets in a block)
   Output Parameters: RECEIVE, AREC, $LW_r$, $UW_r$
Begin
   if not set(RECEIVE(((X-1 div i)-$LW_r$+2) + (X-1 mod i)+1) then
      switch_on(RECEIVE(((X-1 div i)-$LW_r$+2) + (X-1 mod i)+1);
      if (X-1 div i)+1 > $UW_r$ then
      $UW_r$ := (X-1 div i)+1;
    end if;
    if receipt of the packet completes reception of an entire block then
      if (X-1 mod i)+1 = $LW_r$ then
        $LW_r$ := j where j is the second unset bit in AREC;
        copy all bits from the j through L blocks of RECEIVE into the
         first through (L-j+1) blocks of RECEIVE and clear the
         remaining bits;
        copy the j through L elements of AREC into the first
         through (L-j+1) elements of AREC and clear the remaining
         elements;
      else
        switch_on(AREC((X-1 div i)-$LW_r$+1))
      end if;
    end if;
else
    discard data packet;     -- retransmission overlap
end if;
End
```

6. Operation to Evaluate Connection Request Parameters

As part of the three-way handshake connection establishment procedure, the receiver must evaluate the transmitter's recommended values for the negotiated parameters. Because the format of the connection request (*Conn_Req*) message has been left abstract, the algorithm for evaluating it must also be abstract. The purpose of this operation is simply to allow the receiver to either accept, without modification, the transmitter's recommended values, or to modify those parameters which are unacceptable, and return the modified values as part of the connection acknowledgment message (*Conn_Ack).*

```
Operation  Evaluate
  Input Parameters: Conn_Req
  Output Parameters: Conn_Ack

Begin
  for all negotiated parameters in Conn_Req loop
    if $parameter_i$ is not acceptable then
```

```
          modify parameter$_i$;
        end if;

    insert parameter$_i$ into Conn_Ack message;
End
```

7. Operation to Determine Acceptability of Connection Acknowledgment

The transmitter, upon receiving the connection acknowledgment message (*Conn_Ack*) from the receiver, must determine whether a successful negotiation of parameters has occurred. If the values of the parameters returned in the *Conn_Ack* message are acceptable, then a connection confirmation message (*Conn_Conf*) is returned to the receiver and the connection becomes active. Otherwise, depending upon the implementation, the transmitter will either retransmit a connection request message (*Conn_Req*), or transmit a disconnect (*Disc*) message and notify its host of a failed connection attempt.

The operation to determine acceptability of the connection acknowledgment message returns a boolean result to the transmitter.

```
Operation acceptable
   Input Parameters: Conn_Ack
  Output Parameters: boolean result

Begin
  for all negotiated parameters in Conn_Ack loop
     if parameter$_i$ is not acceptable then
        return False;
     end if;

  return True;
End
```

VII. FORMAL SPECIFICATION

In the previous chapters of this thesis, the specification model and the structures used in the specification have been defined and described in detail. In this chapter, the specification structures will be used in a manner similar to programming language library packages. That is, no elaboration of their methods will be revealed. It is assumed that the reader is familiar with the functions and design characteristics of lightweight transport protocols, and in particular with the AT&T Transport Protocol.

The first section of this chapter will describe the specific functions of each of the protocol's FSM's. The second section contains the Variable Initialization Table (VIT) to provide a consolidated view of all of the local and shared variables referenced within the specification. Finally, the last section contains the PAT for the specification.

A. FINITE STATE MACHINE DESCRIPTIONS

Each of the machines in the protocol is solely responsible for performing a specific set of tasks. The machines can be envisioned as executing in parallel, with the caveat that the current state of shared variables may influence transitions between some states. The machines have been designed to be deterministic in order to provide a more precise specification.

1. Machine T1

Machine T1 is responsible for transmitting new data packets, and for retransmitting data packets which belong to unacknowledged blocks whose *count* value has reached zero. Figure 7 depicts the FSM for T1.

T1 begins executing its tasks upon the global variable *T_ACTIVE* being set TRUE by machine T2 subsequent to successful connection establishment. In mode 0, T1 transmits data packets, unchecked by flow control or error recovery, until all data has been transferred. In mode 1, T1 first determines that there is buffer space available to receive additional data packets, and then transmits an appropriate amount of data. This check is made by comparing the last reported value of *buffer_available*, from the receiver control packet, with the current value of *NOU*. Data packets will be transmitted until the number of blocks outstanding equals the amount of buffer space available.

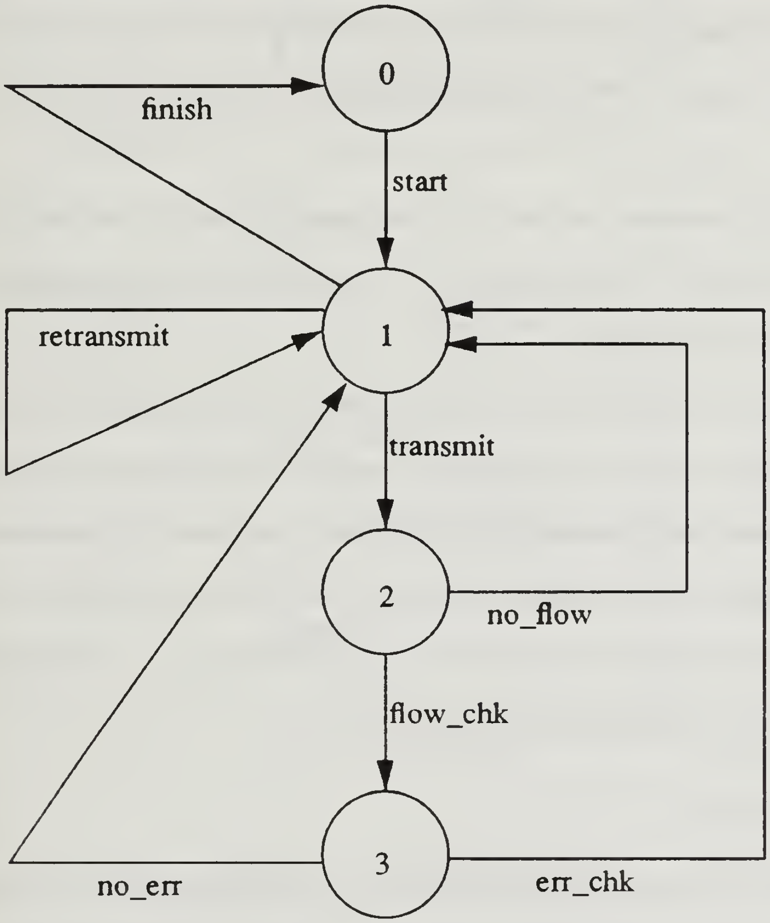


Figure 7 : T1 State Diagram

When operating in mode 2, T1 first determines whether there are any records in the LUP for which the *count* field has reached zero. If so, the necessary packets are retransmitted prior to transmission of any new packets. Retransmission does not require a check of *buffer_available*, as receipt of the retransmitted block will create availability in the reordering buffer by allowing currently buffered blocks to be passed on to the receiving host. As part of the retransmission process, the *count* value for the block is reset.

When all necessary packets have been retransmitted, T1 will begin transmitting new data packets, flow control permitting. Upon transmission of a complete block, T1 will increment *NOU* and UW_t.

Upon either transmission or retransmission of a data packet, the local variable *sent* is set to TRUE. This is an indication to machine T3 that data has been transmitted, and that the control packet retransmission frequency may have to be adjusted. Each time machine T3 sends a control packet, the value of *sent* is reset to FALSE. This interactive toggling is the mechanism which allows the protocol to adjust the retransmission timeout period based upon current logical connection activity.

2. Machine T2

Machine T2 is first responsible for connection establishment, and upon successful connection establishment, for receipt of receiver control packets. T2 activates the execution of machines T1, T3, and T4 upon connection establishment, and also ensures the systematic return to the idle (initial) state of all other transmitter machines upon connection failure or completion of data transfer. The state diagram for T2 is presented in Figure 8.

The transmitter will attempt to establish a connection a predetermined number of times before giving up and reporting to the host that the attempt has failed. The local variable *max_attempts* holds the value for the number of attempts that will be executed.

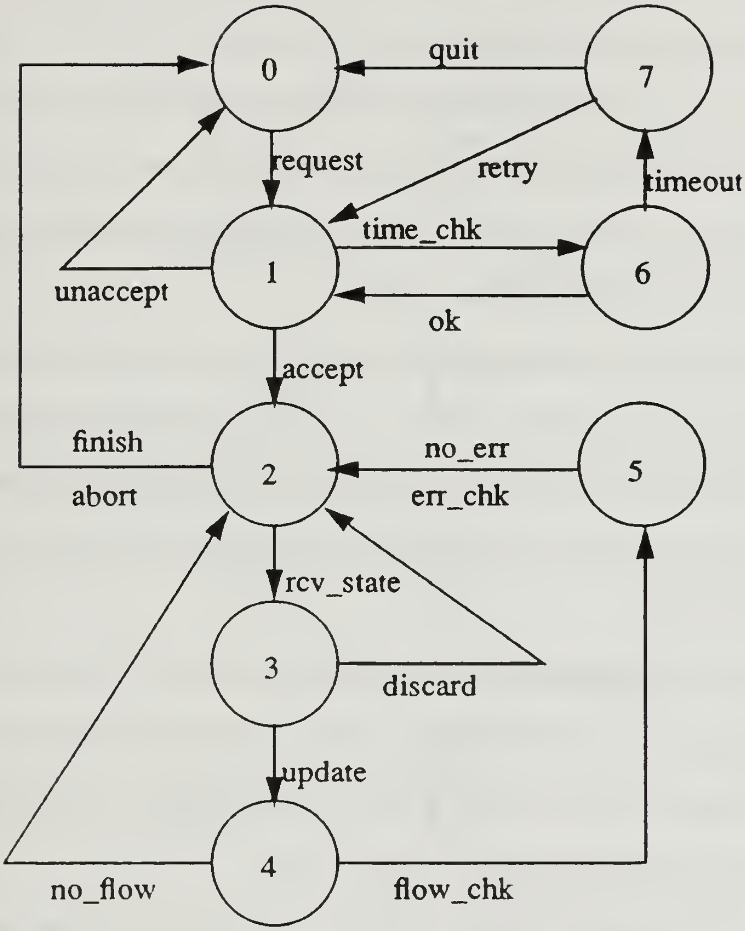


Figure 8 : T2 State Diagram

(An attempt, in this sense, includes the standard three-way handshake).

The event *clock_tick* is used as a pseudo timing mechanism to assist in determining when an attempt is likely to have failed. Each time a *clock_tick* occurs, the variable *delay* is incremented and its value is compared to the variable *reset*, which holds the maximum delay value. If *delay* becomes equal to *reset*, the current connection attempt is assumed to have failed, and the value of the counter *attempts* is incremented. If the value of *attempts* becomes equal to *max_attempts* the connection establishment process is aborted, and the host is notified. If the connection is established, the boolean variable

T_active will be set to TRUE to allow other transmitter machines to begin their execution.

In their idle states, machines T1, T3, and T4 (state 1) are unable to make a transition until *T_active*, which is initially FALSE, becomes TRUE. At this time, T2 transitions into the control packet receiving function, while at the same time it monitors the boolean variable *Disconnect*, which will be set to TRUE by machine T3 upon determination of an unexpected disconnection.

When T2 receives a control packet, it first checks to make sure that the sequence number is higher than the sequence number of the last packet. Only monotonically increasing packet sequence numbers are used to update; out of order packets are discarded. Depending upon the mode of operation, the packet is then used to perform flow and/or error control.

The flow control operation **balance** updates NOU and LW_t based upon the current state of the bit map LOB. The field *buffer_available* from the control packet is also used to perform flow control, as the transmitter first checks this value to determine buffer availability prior to transmitting any new data packets.

In mode 2, the operation **update_LUP** is also executed to either remove packets from acknowledged blocks, or to decrement the *count* value of all unacknowledged blocks. If machines T1 and T2 are in contention for use of the LUP, T2 is given priority in order to prevent superfluous retransmissions.

3. Machine T3

Machine T3 is responsible for the transmission of control packets to the receiver. By monitoring the transmission of data packets through the variable *sent*, T3 is able to adjust the control packet transmission rate according to the transmission rate of data packets. Figure 9 depicts the state diagram for T3.

Upon activation by the TRUE state of *T_active*, T3 executes its function each

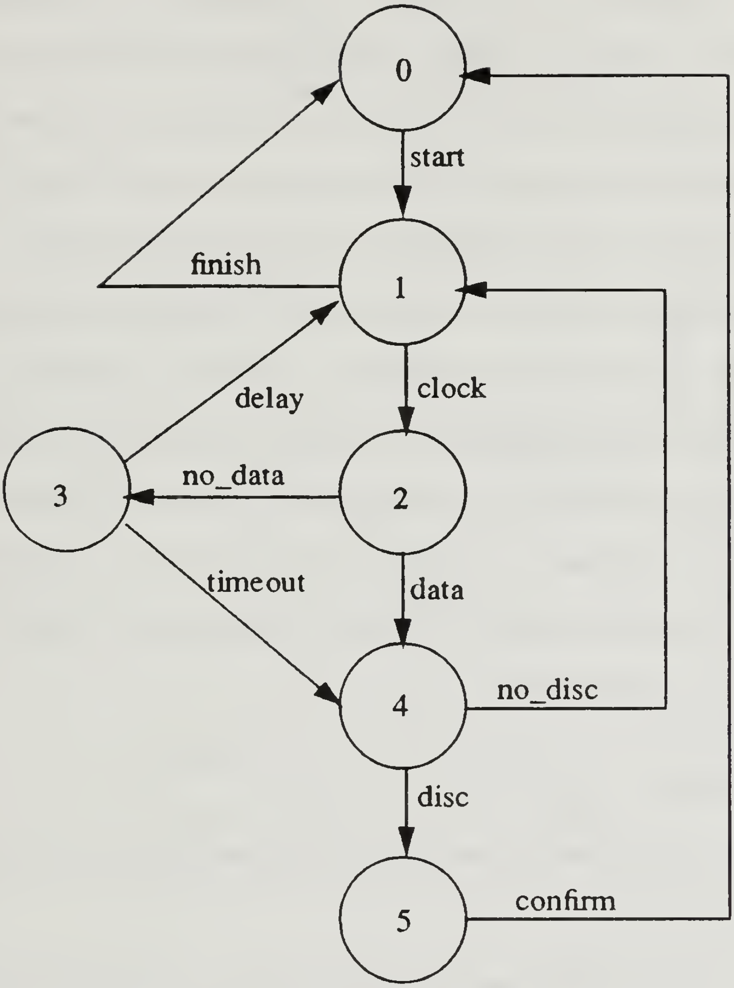


Figure 9 : T3 State Diagram

time the event *clock_tick* occurs. Each time machine T1 transmits a data packets it sets the value of the variable *sent* to TRUE. Alternatively, each time T3 sends a control packet it sets the value of *sent* to FALSE. If *sent* is TRUE when *clock_tick* occurs, then T3 will automatically send a control packet and reset the value of the local counter *k* to 1, and the local counter *count* to 0.

If *sent* is FALSE upon *clock_tick*, then T1 has not sent a data packet subsequent to the last control packet transmission, and T3 must determine whether a control packet transmission is necessary or not. Each time two subsequent control packets are transmitted with no occurrence of a data packet transmission by T1, the value of *k* is doubled, up to a predetermined maximum value *kLim*. Additionally, each time *clock_tick* occurs and the value of *sent* is FALSE, T3 increments the value of *count*. When the value of *count* becomes equal to the value of *k*, a control packet is transmitted.

Another function of T3 is to increment the variable *scount* upon each occurrence of *clock_tick*. The value of *scount* is reset to 0 by machine T2 each time a control packet is received from the receiver. If *scount* ever reaches the predetermined value of *Lim*, then a control packet has not been received from the receiver for *Lim* occurrences of *clock_tick*, and the connection is assumed to have been disrupted. At this point, T3 will set the value of the boolean variable *Disconnect* to TRUE, and machine T2 will begin connection termination procedures.

4. Machine T4

Machine T4 performs all interface with the transmitting host. Its state diagram is shown in Figure 10.

Upon receiving a signal from the host, T4 sets the value of the variable *Transmit* to TRUE. This indicates to machine T2 that a connection should be established. Having set *Transmit*, T4 delays any further action until the variable *T_active* becomes TRUE, indicating that the connection has been established. T4 then deposits the data to be transmitted into the buffer *OUTBUF*, and awaits notification from T2 , through the variable *T_active*, that the transfer has been completed, or the connection has been lost. In either case, T4 notifies the host of the outcome, and returns to state 0.

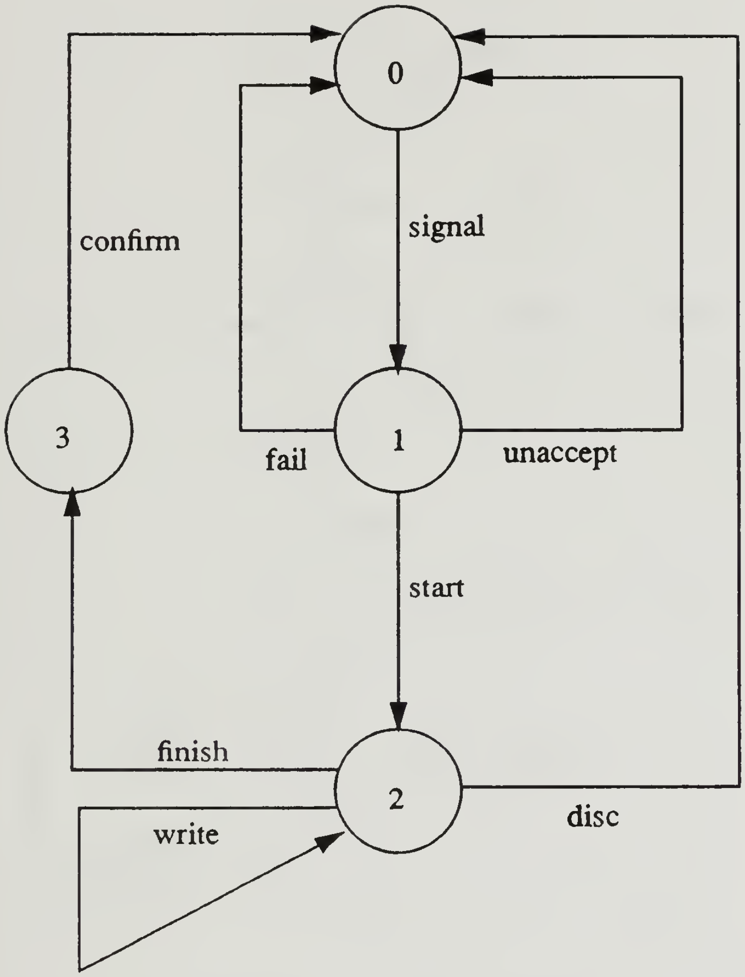


Figure 10 : T4 State Diagram

5. Machine R1

Machine R1 is responsible for receiving data packets from machine T1, and for either buffering the packets or passing them on to the receiving host. The state diagram is depicted in Figure 11.

In mode 0, data packets are passed on to the host with no flow or error control processes being performed. In both mode 1 and mode 2, the operation **Order_insert** is executed to place the packets into their proper sequential order prior to delivery to the

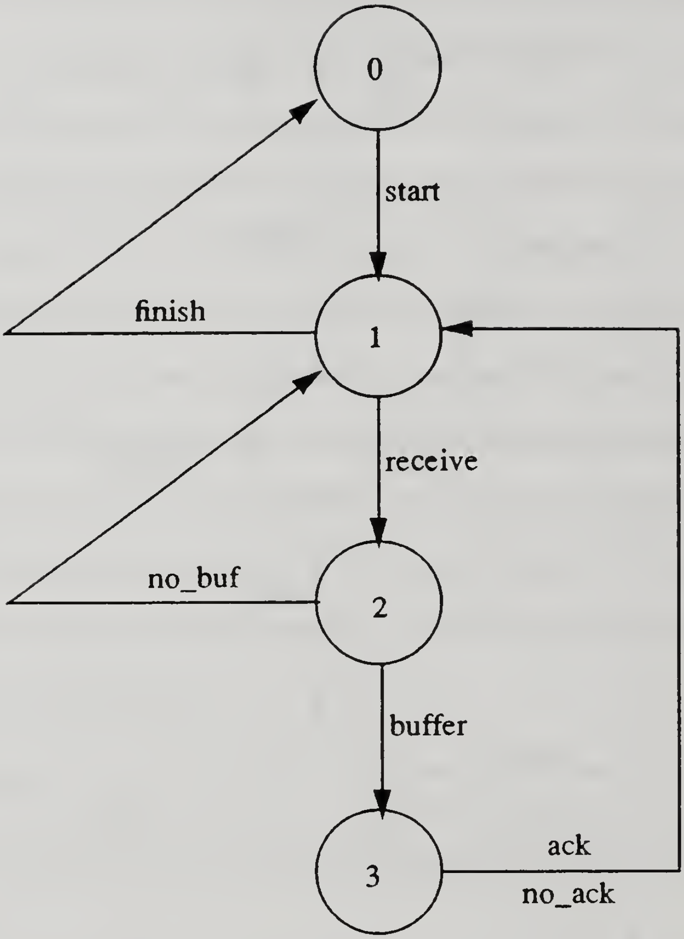


Figure 11 : R1 State Diagram

host. while in mode 2 alone, the additional operation **Process_Packet** is performed. **Order_insert** is a flow control mechanism in which the variables *buffer* (*buffer_available* in the receiver control packet) and *LOB* are updated. **Process_Packet** is an error control process which updates *AREC* and *RECEIVE.*

6. Machine R2

As with machine T2, machine R2 first establishes the connection, and then receives control packets. The state diagram for R2 is shown in Figure 12.

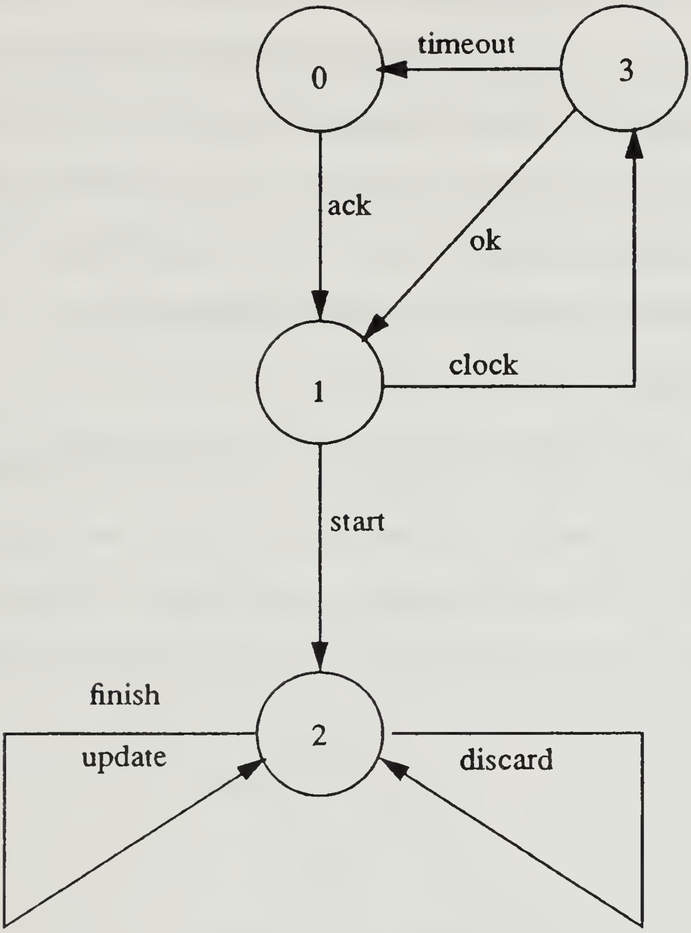


Figure 12 : R2 State Diagram

Upon receipt of a *Conn_Req* message from T2, R2 analyzes the recommended values for the negotiated parameters using the operation **evaluate**, and returns a *Conn_Ack* message to T2, with any modified values for the parameters, if necessary. R2 will then wait for either a *Conn_Conf* message or the first packet of information, either of which will cause it to set the variable *R_active* to TRUE to initiate execution of the other receiver machines. If no message is received from T2 for a predetermined number of *clock_ticks*, then R2 will return to its idle state (state 0) without notifying any of the other machines of the attempt. (It is assumed that if negotiation of the connection parameters machines of the attempt. (It is assumed that if negotiation of the connection parameters failed, T2 will not retransmit a *Conn_Req* message prior to R2 returning to state 0.)

Similar to T2, R2 will reset the value of the local counter *scount* to zero each time a control packet is received from the transmitter. In this particular example, the receiver does not use the information from the transmitter's state packet to update any local variables, though this process could be performed if necessary.

7. Machine R3

Machine R3 is a mirror of machine T3, and as such is responsible for transmission of control packets. The only difference is that the boolean variable *received* is used to monitor the receipt of transmitter control packets in the same way that the variable *sent* was used in machine T3. The state diagram for R3 is shown in Figure 13.

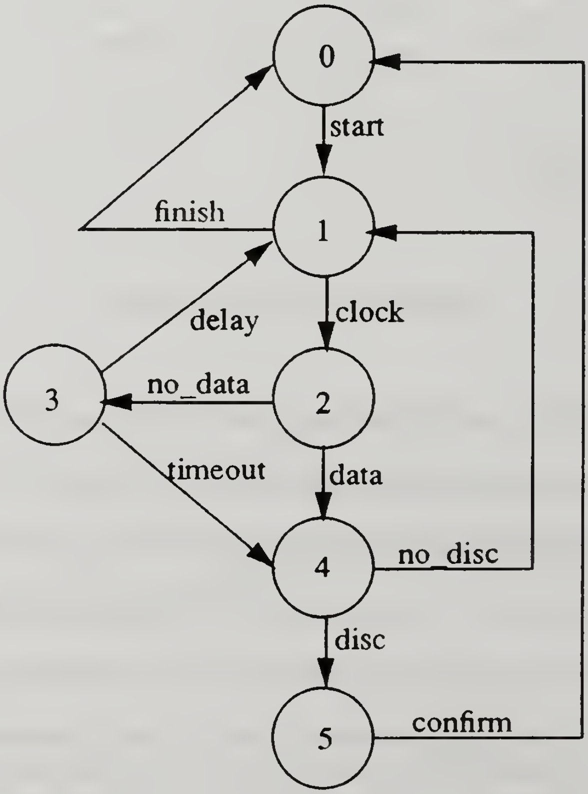


Figure 13 : R3 State Diagram

8. Machine R4

Machine R4 provides the interface to the receiving host by maintaining the reordering buffer, *INBUF*. Upon signal from the host, and availability of the next sequential packet of data, R4 will pass on the data to the host using the operation **Ordered_Retrieval**. When an entire block worth of packets have been removed from *INBUF*, the variable *buffer* is incremented to indicate the additional buffer space available. If the variable *Disconnect* is set TRUE by machine R3, R4 will notify the host of the failed connection. The state diagram for R4 is depicted in Figure 14.

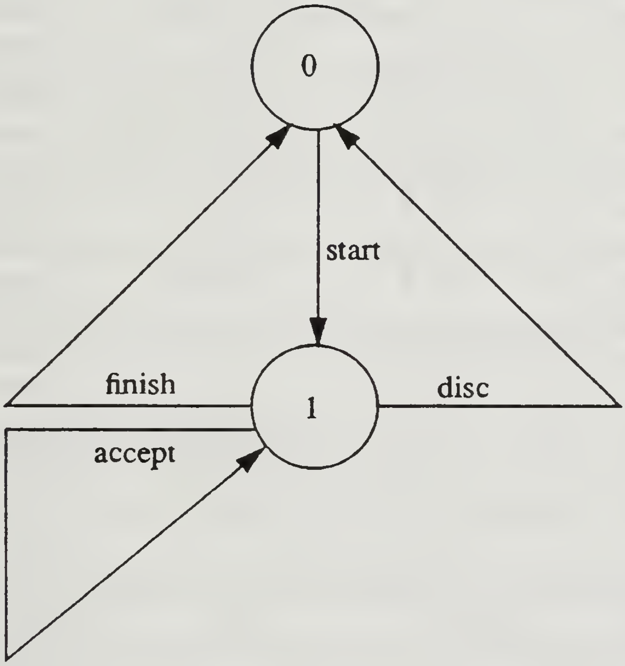


Figure 14 : R4 State Diagram

I. VARIABLE INITIALIZATION TABLE (VIT)

The VIT is presented here to provide a consolidated view of all of the variables referenced within the specification, along with their initial values and the machines which are influenced by them. Influenced, in this sense, means that a machine reads or writes to a particular variable within at least one of its transitions.

TABLE 1: VARIABLE INITIALIZATION TABLE

Name	Type	Range	Initial	Influence	Purpose
Block_size	Constant	---------	TBD	R1,R4	Number of packets per block
Buffer_size	Constant	---------	TBD	R1,R4	Max buffer size at receiver
L	Constant	---------	Buffer_size	T1,T2,R1	Limit on outstanding packets
mode	Enum	0,1,2	TBD	T1,T2,R1,R4	Mode of operation
T_active	Boolean	T, F	F	T1,T2,T3,T4	Execution start and end signal
eot	Boolean	T, F	F	T4	End of transmission indicator
Transmit	Boolean	T, F	F	T2,T4	Transmission request indicator
Accept	Boolean	T, F	T	T2,T4	Connection acceptability indicator
Fail	Boolean	T, F	F	T2,T4	Failed connection indicator
sent	Boolean	T, F	F	T1,T3	Data transmission indicator
attempts	Positive	0 - MAX	0	T2	Connection attempts counter
max_attempts	Constant	---------	TBD	T2	Connection attempts limit
LW_t	Positive	1 - MAX	●	T2	Number of blocks acknowledged
UW_t	Positive	1 - MAX	T	T1	Number of blocks transmitted
NOU	Positive	0 - MAX	0	T1,T2	Number of blocks outstanding
buffer *	Positive	0 - MAX	Buffer_size	T1,T2	Current buffer available at receiver
delay *	Positive	0 - MAX	0	T2	Connection packet time counter

TABLE 1: VARIABLE INITIALIZATION TABLE

Name	Type	Range	Initial	Influence	Purpose
reset *	Constant	---------	TBD	T2	Connection packet timeout value
scount *	Positive	0 - MAX	0	T2,T3	State packet expectation counter
Lim *	Constant	---------	TBD	T3	State packet expectation timeout
count *	Positive	0 - MAX	F	T3	State packet transmission counter
k *	Positive	1 - MAX	1	T3	State packet transmission interval
kLim *	Constant	---------	TBD	T3	State packet interval limit
Disconnect *	Boolean	T, F	F	T2,T3	Disconnection sensor indicator
high *	Positive	0 - MAX	0	T2	Highest block transmitted/ received
R_active	Boolean	T, F	F	R1,R2,R3,R4	Execution start and end signal
received	Boolean	T, F	F	R1,R3	Data reception indicator
LW_r	Positive	1 - MAX	1	R1	Number of blocks acknowledged
UW_r	Positive	1 - MAX	1	R1	Number of blocks transmitted

J. PREDICATE ACTION TABLE (PAT)

The PAT contains all of the information necessary to follow the execution of the protocol. All of the transitions, variables, and operations have been defined in prior sections of this thesis, and the overall function of each machine has been discussed. The PAT, along with the FSM's contained in the first section of this chapter, provides the framework from which either a full or partial analysis can be conducted.

TABLE 2: PREDICATE ACTION TABLE

Transition	Enabling Predicate	Action
	Machine T1	
start	T_active=T	null
finish	T_active=F	null

retransmit	T_active=T ^ mode=2 ^ **expired***(LUP)	enqueue(LUP(*expired packet)*,T_CHAN);
		sent:=T; inc(NOU)

transmit	T_active=T ^ not(empty(OUTBUF))
	^ (mode=0 v ((NOU<L
	^ buffer-NOU>0)
	^ (mode=1 v not(**expired***(LUP)))))
		enqueue(*DATA*,T_CHAN);
		sent:=T; inc(UW_t);

no_flow	mode=0	dequeue(OUTBUF)
flow_chk	mode=1 v mode=2	inc(NOU)
no_err	mode=1	dequeue(OUTBUF)

err_chk	mode=2	insert(OUTBUF(front),LUP);
		dequeue(OUTBUF)

	Machine T2	
request	Transmit=T ^ Accept=T ^ Fail=F	enqueue(*Conn_Req*,T_CHAN)

accept	not(empty(R_CHAN)) ^
	R_CHAN(front)=*Conn_Ack* ^
	acceptable*(R_CHAN(front))
		T_active:=T;
		enqueue(*Conn_Conf*,T_CHAN);
		dequeue(R_CHAN)

unaccept	not(empty(R_CHAN)) ^ R_CHAN(front)=*Conn_Ack* ^ not(**acceptable***(R_CHAN(front)))	Accept:=F; dequeue(R_CHAN)
clock	*clock_tick*	inc(delay)
time_ok	delay<reset	null
timeout	delay=reset	inc(attempts); delay:=0
retry	attempts<max_attempts	enqueue(*Conn_Req*, T_CHAN)
quit	attempts=max_attempts	Fail:=T
finish	Transmit=F ^ empty(OUTBUF) ^ empty(LUP) ^ Disconnect=F	T_active:=F; enqueue(*Disc*,T_CHAN)
abort	Disconnect=T	T_active:=F
rcv_state	not(empty(R_CHAN)) ^ Disconnect=F	null
discard	R_CHAN(front).seq<=high v R_CHAN(front)=*Conn_Ack*	dequeue(R_CHAN)
update	R_CHAN(front).seq>high	scount:=0; high:=R_CHAN(front).seq

TABLE 2: PREDICATE ACTION TABLE

Transition	Enabling Predicate	Action
no_flow	mode=0	dequeue(R_CHAN)

flow_chk	mode=1 v mode=2	**Balance***(R_CHAN(front).LW_r , LW_t , L, LOB, NOU);
		LW_t:=R_CHAN(front).LW_r;
		buffer:=R_CHAN(front).buffer_available;

no_err	mode=1	dequeue(R_CHAN)

err_chk	mode=2	**Update_LUP***(LW_r , LUP, LOB);
		dequeue(R_CHAN)

	Machine T3	
start	T_active=T	null
clock	*clock_tick*	inc(scount)
no_data	sent=F	inc(count)
delay	count<k	null

timeout	count=k	enqueue(*T_state*,T_CHAN);
		k:=min(2*k,kLim)

data	sent=T	enqueue(*T_state*,T_CHAN); k:=1
no_disc	scount<Lim	sent:=F; count:=0
disc	scount=Lim	Disconnect:=T
confirm	T_active=F	null
finish	T_active=F	null
	Machine T4	
signal	*transmission signal from host*	Transmit:=T
fail	Fail=T	Transmit:=F; *notify host of failure to connect*
unaccept	Accept=F	*notify host of unacceptable connection*
start	T_active=T	null
write	not(full(OUTBUF)) ^ not(eot) ^ T_active=T	enqueue(*data stream from host*, OUTBUF)
finish	eot ^ T_active=T	Transmit:=F
confirm	T_active=F	*notify host of completion*
disc	T_active=F	*notify host of disconnect*

TABLE 2: PREDICATE ACTION TABLE

Transition	Enabling Predicate	Action
	Machine R1	
start	R_active=T	null
finish	R_active=F ^ empty(INBUF)	null
receive	not(empty(T_CHAN)) ^ T_CHAN(front)=*DATA*	received:=T
no_buf	mode=0	*pass* T_CHAN(front) *to host*; dequeue(T_CHAN)

buffer	mode=1 v mode=2	**Order_insert***(INBUF, T_CHAN(front), Block_size, Buffer_size);
		dec(buffer); update(LOB)

no_ack	mode=1	dequeue(T_CHAN)

ack	mode=2	**Process_Packet***(RECEIVE, AREC, LW_r, UW_r, L, T_CHAN(front).seq, Block_size);
		dequeue(T_CHAN)

	Machine R2	

ack	not(empty(T_CHAN)) ^ T_CHAN(front)=*Conn_Req*	**evaluate***(*Conn_Req);*
		dequeue(T_CHAN);
		enqueue(*Conn_Ack*, R_CHAN)

clock	*clock_tick*	inc(delay)
time_ok	delay<reset	enqueue(*Conn_Ack*, R_CHAN)
timeout	delay=reset	null

start	not(empty(T_CHAN)) ^ (T_CHAN(front) = *Conn_Conf* v T_CHAN(front) = *(T_state))*	R_active:=T;
		if T_CHAN(front)=*Conn_Conf* then dequeue(T_CHAN); end if;

finish	Disconnect=T ^ not(empty(T_CHAN)) ^ T_CHAN(front)=*Disc*	R_active:=F

update	not(empty(T_CHAN)) ^ T_CHAN(front)=*T_state* ^ T_CHAN(front).seq>high	scount:=0; high:=T_CHAN(front).seq;
		dequeue(T_CHAN)

discard	not(empty(T_CHAN)) ^ (T_CHAN(front)=*Conn_Conf* v T_CHAN(front)=*Conn_Req* v (T_CHANfront)=*T*_state ^ T_CHAN(front).seq<=high))	dequeue(T_CHAN)

TABLE 2: PREDICATE ACTION TABLE

Transition	Enabling Predicate	Action

lost_ack	not(empty(T_CHAN)) ^ T_CHAN(front)=*Conn_Req*	dequeue(T_CHAN);
		enqueue(*Conn_Ack*, R_CHAN)

	Machine R3	
start	R_active=T	null
clock	*clock_tick*	inc(scount)
no_data	received=F	inc(count)
delay	count<k	null

timeout	count=k	enqueue(*R_state*,R_CHAN);
		k:=min(2*k,kLim)

data	received=T	enqueue(*R_state*,R_CHAN); k:=1
no_disc	scount<Lim	received:=F; count:=0
disc	scount=Lim	Disconnect:=T
confirm	R_active=F	null
finish	R_active=F	null
	Machine R4	
start	R_active=T ^ (mode=1 v mode=2)	null

accept	Disconnect=F ^ not(empty(INBUF)) ^ *signal from host*	**Ordered_Retrieval***(INBUF, Buffer_size, Block_size);
		inc(buffer); update(INBUF)

finish	R_active=F ^ empty(INBUF) ^ Disconnect=F	null
disc	Disconnect=T	*notify host of disconnect*

VIII. ANALYSIS

A. SYSTEM STATE ANALYSIS

The *system state analysis* for the connection establishment phase of the protocol is shown in Figure 15. System state analysis is a form of reachability analysis that reduces the total number of states explored (which can easily become unmanageable). A discussion of its uses and limitations is given in [LUND91].

In Figure 15, the notation (X,Y) represents a partial system state with machine T2 in state X and machine R2 in state Y. The other six machines are not shown because only these two are involved in connection establishment.

The contents of the communication channels (shared variables) and the values of the local variables are not shown in the diagram, for the sake of brevity. These were figured into the analysis and should be considered as part of any detailed analysis.

The connection analysis begins with both T2 and R2 in state 0, which is system state (0,0). The process is initiated when a connection request is received by the transmitter, indicated by the *signal* transition being taken by T4. (The reader may confirm by checking the predicate action table (Table 2) for machine T2; specifically, the enabling predicate for the *request* transition). This transition leads to system state (1,0).

In state (1,0), either the *ack* transition by R2 or the *clock* transition by T2 are possible. The *clock* transition is a part of T2's timing process, and will eventually lead to a timeout and failure to connect if R2 does not respond. The *ack* transition is taken by R2 upon receipt of the *Conn_Req* message from T2, leading to system state (1,1).

If the connection establishment is successful, the process will lead to system state (2,2) or (2,0), and the data transfer phase will be entered. Unsuccessful attempts lead back to state (0,0).

The connection establishment analysis shows a total of 32 system states; 13 of these are duplicates, so there are 19 unique system states. The analysis shows that the connection establishment will be completed without deadlock (if the physical link is up), or that the system will return to the initial system state without deadlock.

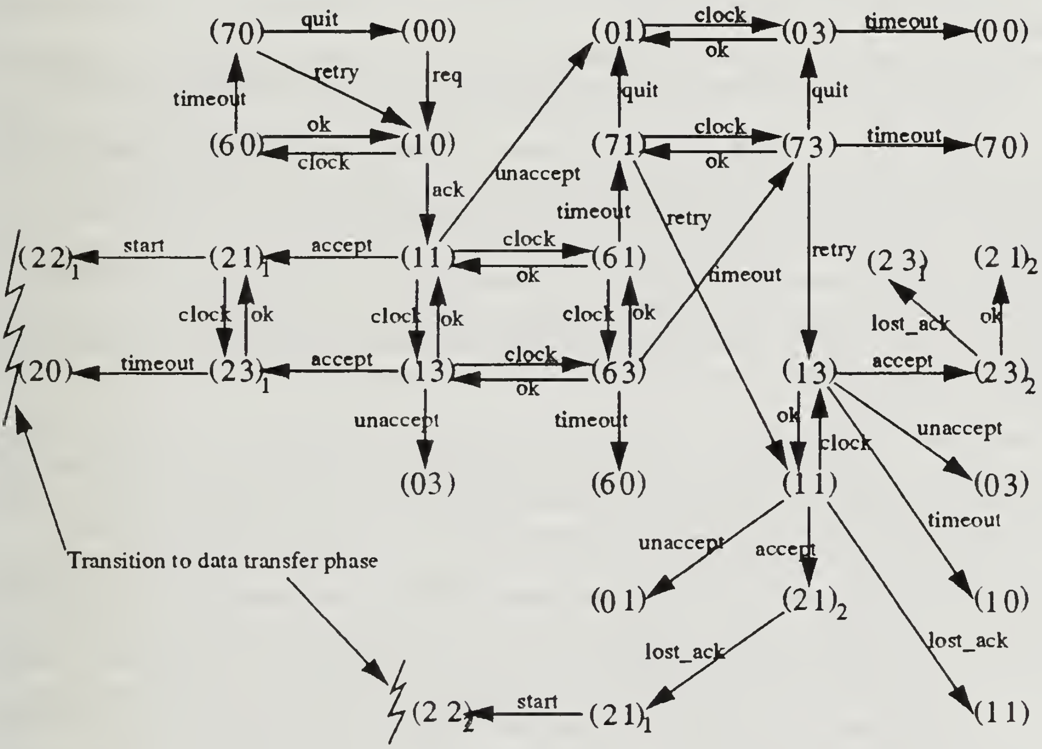


Figure 15 : Connection Establishment Analysis
(The pair (X,Y) represents state X of machine T2 and state Y of machine R2)

B. FLOW CONTROL AND ERROR RECOVERY ANALYSIS

1. The Problem

As discussed earlier in this thesis, there are inherent problems with combining flow control and error recovery mechanisms by overloading variables. The most common result of this practice is that transmission of new data packets is halted pending recovery of a lost packet, even when buffer space is available to accommodate additional packets. The high bandwidth available with fiber optic transmission media only exacerbates this problem.

In this section, the flow control and error recovery mechanisms of the AT&T Protocol will be analyzed in detail. Basing network parameters upon values used in the performance analysis section of [NETR90], it will be shown that large delays may be experienced in data packet transmission. Because most of the flow control problems encountered relate directly to the window acknowledgment mechanism, a brief discussion of its use is appropriate.

For any system in which ARQ acknowledgment is used, the minimum amount of time the transmitter must wait before retransmitting a packet is the RTD plus the processing time of the receiver. This is the amount of time the transmitter would expect to wait before receiving an acknowledgment of a packet. Normally, an additional "fudge factor" is incorporated into the timeout period to allow for variances in RTD. Based upon research discussed in [NETR90], a retransmission timeout of two to three times the RTD is recommended by the authors.

As the window size increases, the transmitter is able to increase the amount of data in transit to the receiver, and thus increase throughput. The problem with large windows is that an equivalent amount of space must be maintained at the transmitter and receiver to buffer the data packets. In implementations with multiple logical connections to a single machine, this can be costly in terms of storage requirements and access times.

In this protocol, the error recovery mechanism is bound on the lower side by the RTD, and on the upper side by the window size. In accordance with the results in [NETR90], a retransmission timeout of two times the RTD, and a window size of 1024 packets will be used. A one-way delay of 15 ms is arbitrarily chosen, resulting in a retransmission timeout of 60 ms. Because the *count* field in the LUP is based upon control packet transmission rates, and a transmission rate of 3 ms is assumed initially, the retransmission timeout period is converted to 20. Recall that this value will be decremented by *k*, the current control packet transmission rate, each time a receiver control packet is received.

In the analysis which follows, it will be assumed that a perfectly efficient system exists in which processing time at the receiver and transmitter is instantaneous. Figure 16 shows the timing diagram which will be used as a reference for the following discussion.

At time 0 ms the transmitter begins transmitting data packets at a rate of 20k per second (20 per ms). *Assume that the first packet is lost.* The transmitter continues to transmit at this rate until forced to stop by some other mechanism within the protocol, i.e. until *buffer_available - NOU* = 0. So, every 15 ms, 300 packets are transmitted.

At 30 ms, 600 packets have been transmitted, but due to the RTD the transmitter is just receiving the first receiver control packet indicating that LW_r is 1, and *buffer_available* is 1024. *NOU* at this point is 600. When the transmitter calculates buffer availability it determines that 424 more packets may be transmitted.

At 36 ms, the receiver sends the control packet indicating an LW_r equal to 1 (the first packet was lost), an *LOB* indicating that packets 2 through 420 have been buffered, and a *buffer_available* of 605 (1024 - 419). This control packet will be received by the transmitter at 51 ms.

In addition to receiving the above control packet at 51 ms, the transmitter knows

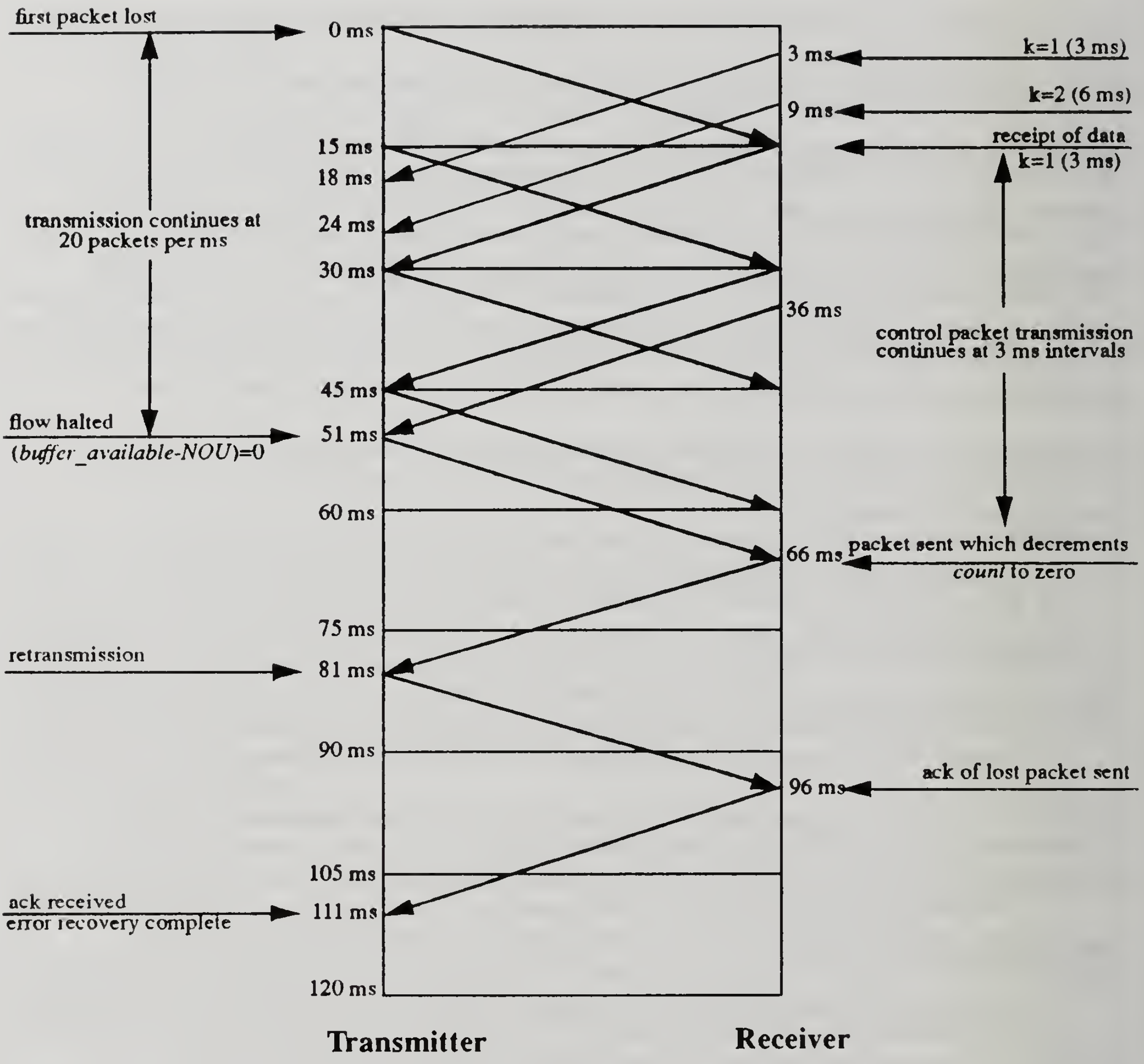


Figure 16 : Data Transfer Timing Diagram

that it has transmitted 1020 packets, thus it calculates that *NOU* is equal to 601 (1020 - 419). The flow control equation *buffer_available - NOU* indicates that four more packets may be transmitted, which the transmitter accomplishes in the next few microseconds, bringing the flow control equation equal to zero and halting transmission of further data packets.

Meanwhile, at 3 ms the first receiver control packet was transmitted, with a *k* value of 1 (equivalent to 3 ms). The value of *k* was subsequently doubled to 2, and the next control packet was transmitted at 9 ms. Again, because no data had been received yet, the value of *k* was doubled, postponing transmission of the third control packet to 21 ms, or receipt of the first data packet.

The result of the transmitter's update process of these first two receiver control packets (which were received at 18 and 24 ms, respectively) was that the *count* value for block 1 (packet 1) was decremented by 3 (1 + 2), leaving a value of 17. Because the first data packet was received by the receiver at 15 ms, and data packets continued to arrive until 66 ms, all subsequent receiver control packets contained a *k* value of 1, and were transmitted every 3 ms. Thus, 17 control packets (51 ms) would be needed to decrement *count* to zero and prompt a retransmission of packet 1. The 17th packet would be transmitted by the receiver at 66 ms and received by the transmitter at 81 ms. The transmitter will thus be idle for 30 ms (51 ms through 81 ms), waiting for a shift of the window, or an indication of an error.

At 81 ms the transmitter retransmits block 1 (packet 1). Still, it must wait for a full RTD before it receives acknowledgment from the receiver, and is able to begin transmission of additional data packets. A total of 60 ms (51 ms through 111 ms) has thus been spent in an idle state, waiting for recovery of the first data packet. Put another way, 60 ms of the first 111 ms was spent idle; well over 50% of the total time.

Given that flow control and error recovery are integrated in this protocol, the only way to improve the performance is to recover the lost packet more quickly. In this example, the buffer space available at the receiver was full, preventing additional data packets from being transmitted. It is apparent though, that even with a buffer size of two megabytes, the transmitter would have been forced to halt transmission of new data packets due to the window size of 1024 packets. Even though the additional one megabyte of buffer space would have been available, the *LOB* length would not have provided visibility of any packets beyond the first 1024. Thus, flow control would have been constrained by recovery of the error.

2. The Cause

The problem outlined in the previous section can be traced to an overloaded variable, *NOU*. The only variables that the transmitter checks prior to transmitting new data packets are *buffer_available* and *NOU*. Thus, one would assume that both of these variables should be restricted to accomplishing flow control. However, a closer look reveals a hidden relationship.

The variable *buffer_available* is received as part of the receiver control packet, and is dependent only upon the current status of the reordering buffer. In other words, it is independent of the status of data packets which are enroute from the transmitter.

NOU, on the other hand, is updated upon receipt of the receiver control packet by the *LOB* bit map. *LOB* is limited by the current window position, which is determined by the value of the parameter LW_r. The value of LW_r is determined by the block number containing the lost data packet. It is plain to see that *NOU* is directly influenced by the lost packet, and thus flow control itself is also directly influenced.

The solution to the problem includes separating the variables used for flow control from those used for error recovery, updating *NOU* without the use of *LOB*. The formal specification of this solution is beyond the scope of this thesis, but could involve an error recovery mechanism such as the one outlined in [CLAR88], in which the receiver sends a list of missing packet sequence numbers to the transmitter upon the transmitter's request.

A more obvious problem with this protocol is that no mechanism exists to accomplish rate control. The transmission of data packets is a binary process, that is, it is either “on” or “off”. The transmitter is either sending packets at the negotiated bandwidth, or not at all. There is no provision for the transmitter to adjust the frequency of transmission based upon the current capability of the receiver to receive them. Again, separation of the flow control process could allow a more efficient synchronization between the transmitter and the receiver.

3. An Improved Method

Given that flow control and error recovery are integrated in this protocol, what changes could be made to circumvent the problem discussed in section 1? The solution to this problem is predicated upon the transmitter becoming aware of the lost packet as soon as possible. The only means available to the transmitter for gaining this insight is the *LOB* bit map. If the transmitter could use the bit map to determine the probability of a lost packet, it could initiate error recovery sooner.

Because the first element of the bit map represents the lost block (packet), it will always be a zero. As blocks with sequence numbers greater than the one containing the lost packet are received, the subsequent elements in the bit map will be set to one. Eventually, a pattern of a zero followed by several ones will appear. If the transmitter was to recognize a predetermined pattern, it could take action to recover the missing element.

How many ones should appear before the transmitter initiates error recovery? In other words, how long should the transmitter wait before assuming that a packet is lost? Although this question may sound like a lead in to a timer based solution, there is a major difference. The transmitter knows that a number of blocks that were transmitted after the lost packet have been received, therefore the variance in RTD due to network routing is statistically averaged over a larger sample of packets. It is unlikely that a packet is experiencing a major network delay if all of the packets around it are not experiencing the same delay. This evidence should strengthen the transmitter’s assumption that the packet has indeed been lost.

For the network parameters used in the first section of this chapter, a delay of 10 ms would seem to be enough to indicate the possibility of a lost packet. Figure 17 shows a modified timing diagram, assuming that the transmitter will recognize a pattern of a zero followed by 25 ones as a retransmission catalyst. The number 25 was chosen because in a 10 ms period representing the network delay, 200 packets, or 25 blocks, would have been received by the receiver.

As illustrated in the modified timing diagram, recovery of the error using the pattern recognition catalyst method occurred at 70 ms, thus decreasing the amount of transmitter idle time from 60 ms to 19 ms (over 66%). Additionally, the transmitter does not need to maintain a retransmission counter or timer, except as a backup. By using LW_r to determine the sequence number of the block containing the lost packet, and *LOB* to recognize the catalyst pattern, the transmitter is freed from timely LUP Table updates. If the block size was based upon a standard bit length computer word, then the pattern recognition process could be implemented in hardware, thus further increasing the efficiency of error recovery.

C. DATA FLOW ANALYSIS

Figure 19 contains a partial system state analysis showing the correct behavior of the protocol for the transmission and acknowledgment of one block of data (eight packets). The analysis assumes error-free operation in mode 2 and is included to illustrate the steps required to accomplish data transfer. Network parameters for this section remain the same as for the previous section, as do the assumptions concerning instantaneous processing at each communicating entity. The analysis begins at the system state immediately following successful connection establishment and includes all eight machines.

The notation *(ABCD-WXYZ)* represents the system state with machine T1 through T4

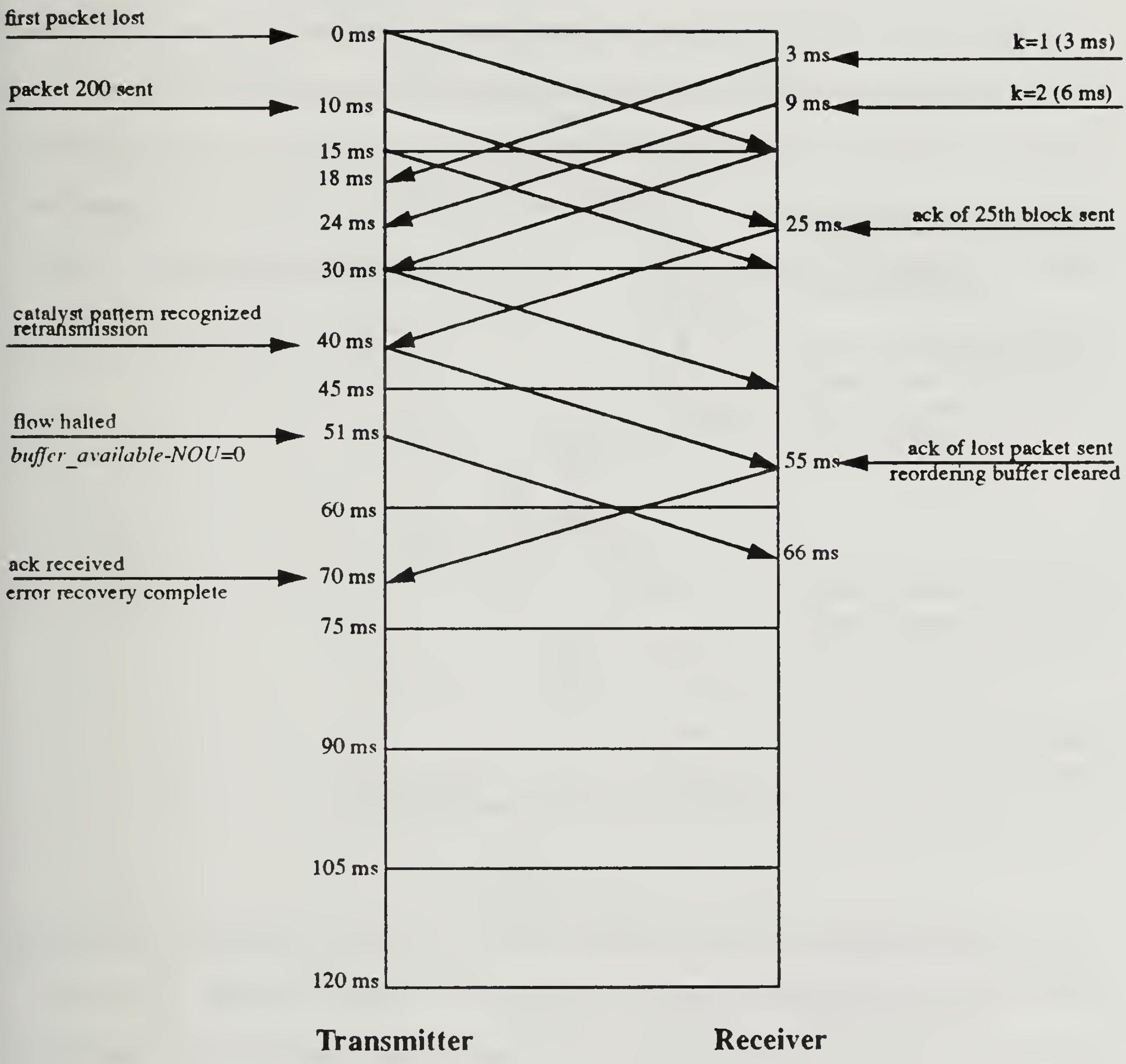


Figure 17 : Modified Data Transfer Timing Diagram

is states *A* through *D*, and R1 through R4 in states *W* through *Z*. In some places, the system state has been divided to separate the transmitter machines from the receiver machines in order to show independent processes. Where this occurs, a subscript ($_T$ or $_R$) will be used to clarify which entity is represented by the partial system state.

Additionally, the analysis represents a *single path* from beginning to end. Divisions in the path represent actions that occur simultaneously and independently, and are enclosed in dashed boxes to "encapsulate" the processes. All paths within a box *must* be taken prior to exiting the box. For example: in Figure 18 path *A* leads into the dashed box at point *x*. Before traversing the path to point *y* outside of the box, paths *B* and *C* must *both* be traversed.

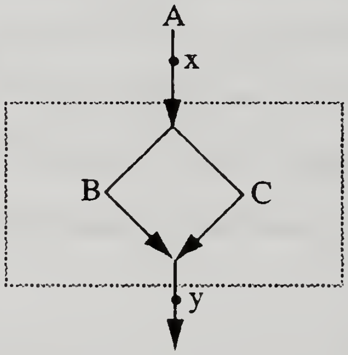
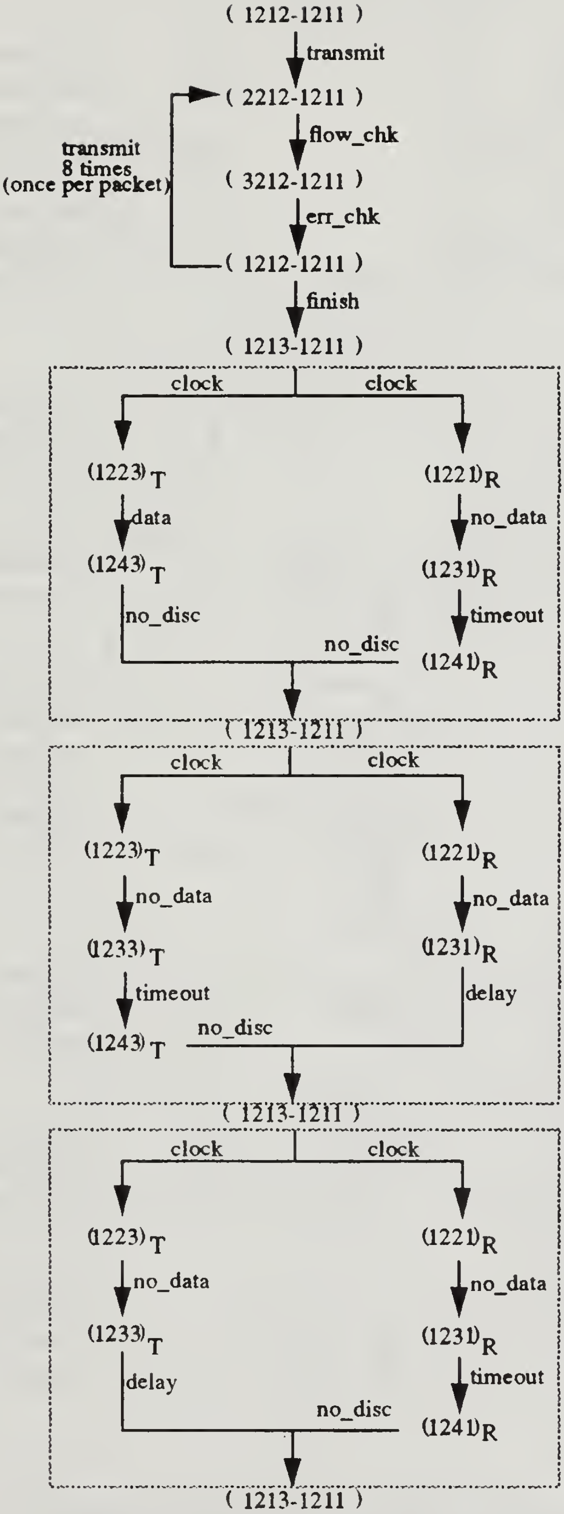


Figure 18 : Path Traversal Notation

The elapsed time is displayed to the left of the system state path and notes concerning actions inherent to certain transitions are displayed to the right of the path. These notes provide a convenient means of tracing the data transfer phase through the predicate action table, as they refer to the machines and transitions which are involved at each step in the process.

TIME **NOTES**



0 ms

Transmitter sends eight packets (one block).

3 ms

Transmitter sends control packet because *sent* is true in machine T3's *data* transition.

Receiver sends control packet because *count=k* in *timeout* transition of machine R3. R3's *k* value is doubled to 2.

6 ms

Transmitter sends control packet because *count=k* in *timeout* transition of machine T3. T3's *k* value is doubled to 2.

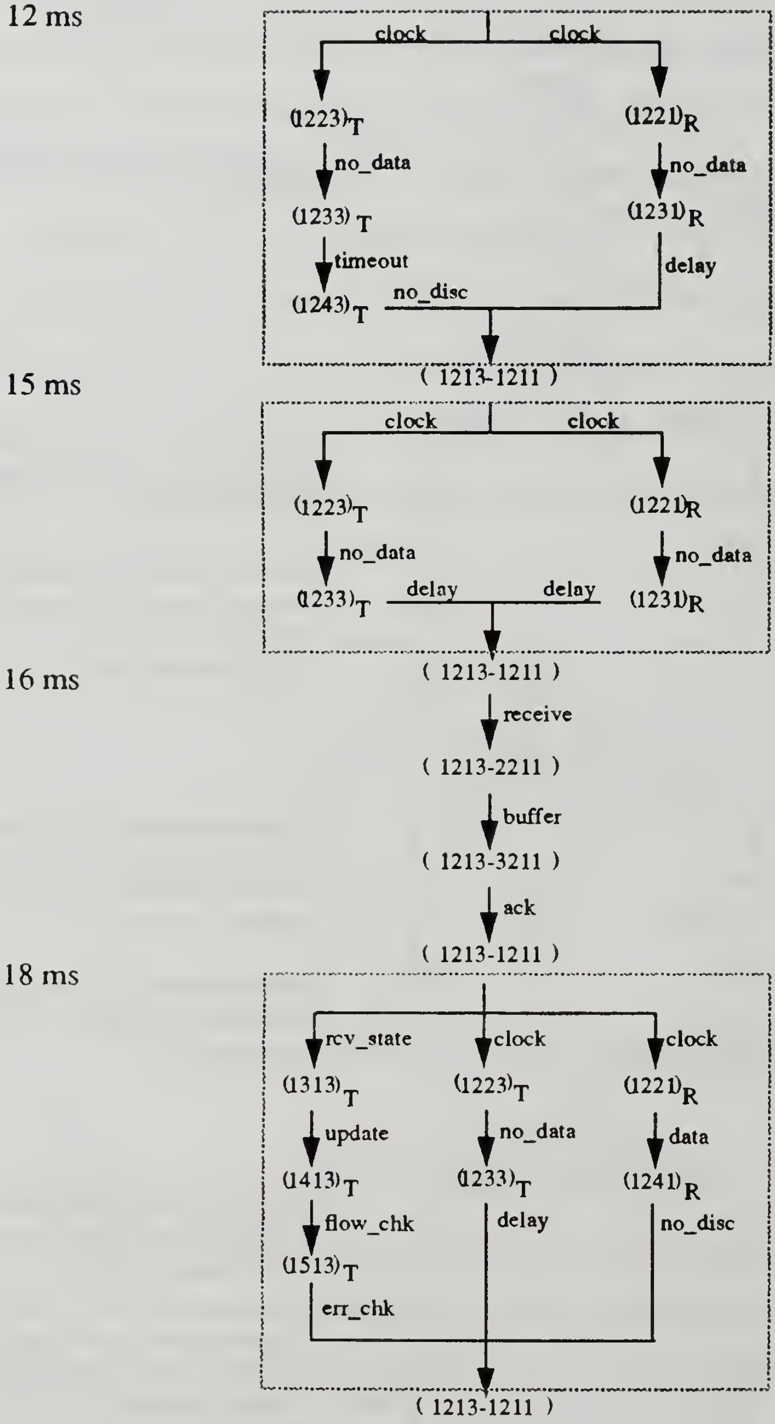
Receiver takes *delay* transition in machine R3: does not send control packet.

9 ms

Transmitter takes *delay* transition in machine T3; does not send control packet.

Receiver sends control packet because *count=k* in *timeout* transition of machine R3. R3's *k* value is doubled to 4.

Figure 19 : Data Transfer Analysis



Transmitter sends control packet because *count=k* in *timeout* transition of machine T3. T3's *k* value is doubled to 4.

Receiver takes *delay* transition in machine R3; does not send control packet.

Transmitter takes *delay* transition in machine T3; does not send control packet.

Receiver takes *delay* transition in machine R3; does not send control packet.

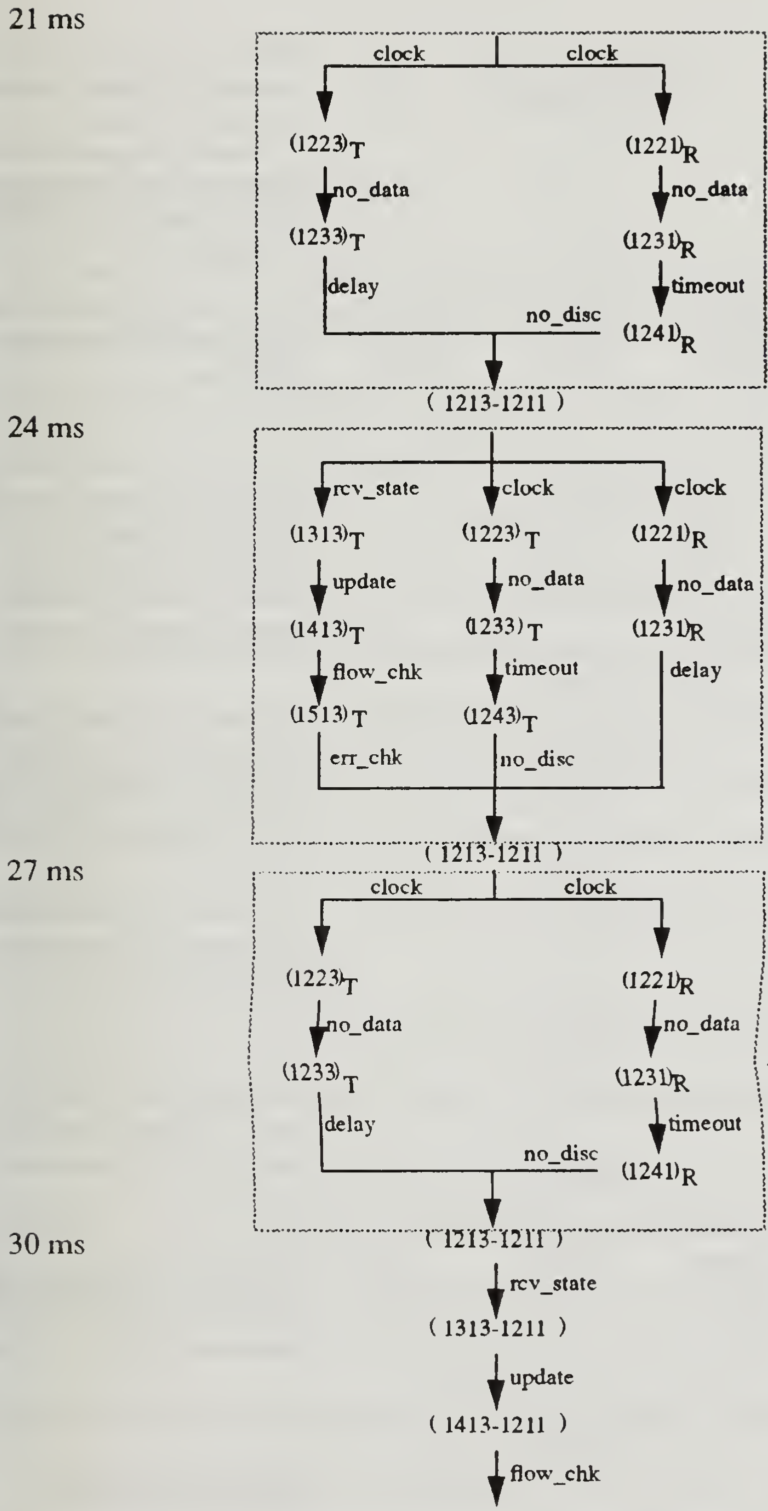
Receiver receives the block of data, buffer's it, and updates its communication structures and variables. The *received* variable is set true by machine R1. The next control packet will reflect receipt of the data.

Transmitter receives first receiver control packet, machine T2 updates as necessary.

Transmitter takes *delay* transition in machine T3; does not send control packet.

Receiver sends control packet because *received* is true in *data* transition of machine R3. R3's *k* value is reset to 1.

Figure 19 Cont'd : Data Transfer Analysis



Transmitter takes *delay* transition in machine T3; does not send control packet.

Receiver sends control packet because *count=k* in *timeout* transition of machine R3. R3's *k* value is doubled to 2.

Transmitter receives second receiver control packet, machine T2 updates as necessary.

Transmitter sends control packet because *count=k* in *timeout* transition of machine T3. T3's *k* value remains 4 (assuming *kLim* is set to 4).

Receiver takes *delay* transition in machine R3; does not send control packet.

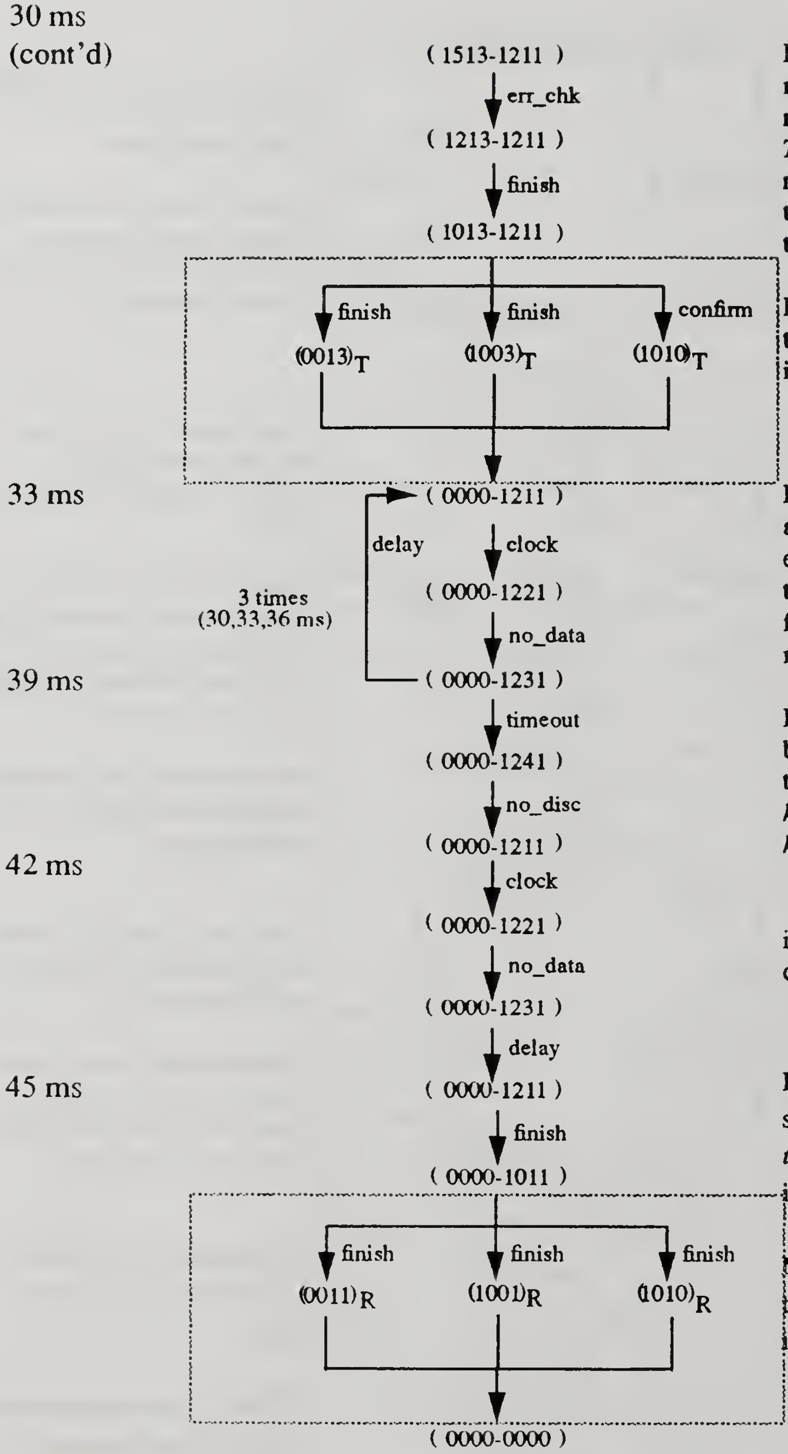
Transmitter takes *delay* transition in machine T3; does not send control packet.

Receiver sends control packet because *count=k* in *timeout* transition of machine R3. R3's *k* value is doubled to 4.

Transmitter receives control packet which acks the data block.

Machine T2 updates the control packet sequence number, NOU and *buffer*.

Figure 19 Cont'd : Data Transfer Analysis



Machine T2 updates the LUP, removing all entries (assuming no losses). T2 also sets *T_Acive* to false, sends a *Disc* message to the receiver to initiate disconnection, and returns to its idle state.

Machines T1, T3, and T4 take their final transitions to their idle states.

Receiver continues to function, as the *Disc* message is enroute. During the 15 ms it takes to receive the message, four *clock* events occur, three result in delays.

Receiver sends control packet because *count=k* in *timeout* transition of machine R3. R3's *k* value remains 4 (assuming *kLim* is set to 4).

Receiver takes *delay* transition in machine R3; does not send control packet.

Receiver receives *Disc* message. Machine R2 sets *R_Active* to false and returns to its idle state.

Machines R1, R3, and R4 take their final transitions to their idle states.

Data transfer complete.

Figure 19 Cont'd : Data Transfer Analysis

IX. CONCLUSION

A. SUMMARY OF RESEARCH

The objective of this thesis has been to demonstrate the requirement for a new generation of transport protocols, and to present a formal specification of the high speed transport protocol introduced in [NETR90]. Fiber optic technology has literally redefined the functional specification for transport protocols, and by examining the problems of current transport protocols the new design approach becomes more clear.

This thesis has covered in detail the design considerations of each of the transport protocol functions: connection management, packet acknowledgment, flow control, and error detection and recovery. The difference between old and new mechanisms for implementing these functions was discussed to demonstrate the increased performance potential provided by transport protocol redesign. The survey of state of the art transport protocol research projects provided a frame of reference for the detailed investigation of one particular protocol design.

The system of communicating machines model provides the framework with which to conduct formal specification of protocols. Using the SCM model, the specification presented in this thesis was developed and an analysis of some select functions was performed. Though a complete analysis of the protocol was not accomplished, the specification provides the base point from which to conduct either a full or partial analysis. A full system state (reachability) analysis of the protocol may prove to be somewhat prohibitive however, as the state space explosion problem will be encountered.

The alternative chosen for this thesis was to use timing diagrams to analyze the behavior of the protocol under realistic network conditions. As demonstrated, the protocol has the potential to suffer significant transmission delays due to the integration of flow control and error recovery mechanisms. An improved error recovery routine was offered, though it still uses overloaded variables for flow and error control. Indeed, no optimal solution may exist for optimizing flow control if it is explicitly or implicitly tied into error recovery.

B. FURTHER RESEARCH OPPORTUNITIES

Many opportunities exist to expand upon the foundation which has been laid with this thesis. First, a redesign of the protocol may be undertaken to separate the mechanisms for performing flow control and error recovery. Using the ideas already incorporated into other prototype lightweight transport protocols, true rate control could be designed into this protocol. Also, a more efficient method of error recovery, using the theory of frequent and periodic exchange of state which this protocol is built around, could be integrated into the design. One interesting method of acknowledgment is to have the receiver control the retransmission of packets which it expects are lost, based upon the prenegotiated inter-transmission time.

It would be interesting to develop a protocol analyzer which specifically dealt with reducing the state space explosion problem by incorporating a set of heuristics for selecting one of many enabled transitions. Because of the highly independent nature of the eight machines within the protocol presented in this thesis, some form of state space reduction will be necessary to accomplish a complete analysis.

An alternative method of analysis for this protocol would be to use graph theory to develop equations which represented the behavior of the various elements of this protocol. For example, the transmission rates of control packets is based upon a clearly established, and predictable, pattern. Additionally, the relationship between data packet and control packet transmission is also clearly established by formula. If an equation was developed to represent the transmission of control and data packets, then predictions of future system states could be projected to influence current transitions. This method might also reduce the state space of the analysis by introducing the theory of *smart states*, which actually represented a future series of transitions.

Because of the modular nature of this protocol, it seems that it would be a rather straightforward problem to develop an object oriented simulation model. Using the simulation, various network parameters could be tested and analyzed quickly and inexpensively, thus helping to develop an optimal version of the protocol. If each of the machines, the transmission media, and the negotiated parameters were designed as separate objects (classes or generic packages), then the opportunities for simulating many different environments would be unbounded. Different versions of many protocol layers could be developed and tested with the model, thus increasing the return on invested time, and increasing student and faculty interest in the field of high speed networks. The existence of a simulation program would also alleviate the requirement for an installed fiber optic network lab.

Fiber optic networks will soon become the standard not only for state of the art communication systems, but for everyday applications such as cable television and the public telephone system. To take advantage of the potentials offered by fiber optics, it will be necessary to implement more effective transport protocols using some of the design concepts discussed in this thesis. As with any other developing technology, the ability to apply new theories to dynamically evolving environments will determine success or failure. This thesis has presented a picture of the current operating environment of high speed networks and has set the stage for a new generation of transport level protocol implementations.

REFERENCES

[CHER86] Cheriton, D.R., "VMTP: A Transport Protocol for the Next Generation of Communication Systems," *SIGCOMM '86 Symposium*, vol. 16, #3, 1986.

[CHES88] Chesson, G., "Xpress Transfer Protocol (XTP) - A Tutorial," *Computer Communications Review*, vol. 20, #5, October 1990.

[CLAR88] Clark, D.D., Lambert, M., and Zhang, L., "NETBLT: A Bulk Data Transfer Protocol," *SIGCOMM '87 Symposium*, vol. 17, #5, 1987.

[DOER90] Doeringer, W.A. and others, "A Survey of Light-Weight Transport Protocols for High Speed Networks," *IEEE Transactions on Communications*, vol. 38, #11, Nov 1990.

[LAPO91] LaPorta, T.F., Schwartz, M., "Architectures, Features, and Implementation of High-Speed Transport Protocols," *IEEE Network Magazine*, May 1991.

[LUND88] Lundy, G.M., Miller, R.E., "A Variable Window Protocol Specification and Analysis," *Protocol Specification, Testing, and Verification, VIII,* North-Holland, 1988.

[LUND91] Lundy, G.M., and Miller,R.E., "Specification and Analysis of a Data Transfer Protocol Using Systems of Communicating Machines," *Distributed Computing*, Dec 1991.

[NETR90] Netravali, A., Roome, W., and Sabnani, K., "Design and Implementation of a High Speed Transport Protocol," *IEEE Transactions on Communications,* vol.38, #11, Nov 1990.

[ROTH92] Rothlisberger, M.J., "Automated System State Analysis," Master's Thesis, U.S. Naval Postgraduate School, to be published in September, 1992.

[STAL88] Stallings, W., *Data and Computer Communications, 2nd ed.*, Macmillan Publishing Co., 1988.

[ZHAN86] Zhang, L. "Why TCP Timers Don't Work Well," *Proc. ACM SIGCOMM Symposium on Communications, Architectures, and Protocols,* Stowe, VT., 1986.

INITIAL DISTRIBUTION LIST

Defense Technical Information Center 2
Cameron Station
Alexandria, VA 22304-6145

Dudley Knox Library 2
Code 52
Naval Postgraduate School
Monterey, CA 93943

Dr. G.M. Lundy, Code CSLn 2
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943

Captain Robert C. McArthur 3
Marine Corps Tactical Systems Support Activity
Camp Pendleton, CA 92055-5080

Commandant of the Marine Corps 1
Code TE 06
Headquarters, U.S. Marine Corps
Washington, D.C. 20380-0001